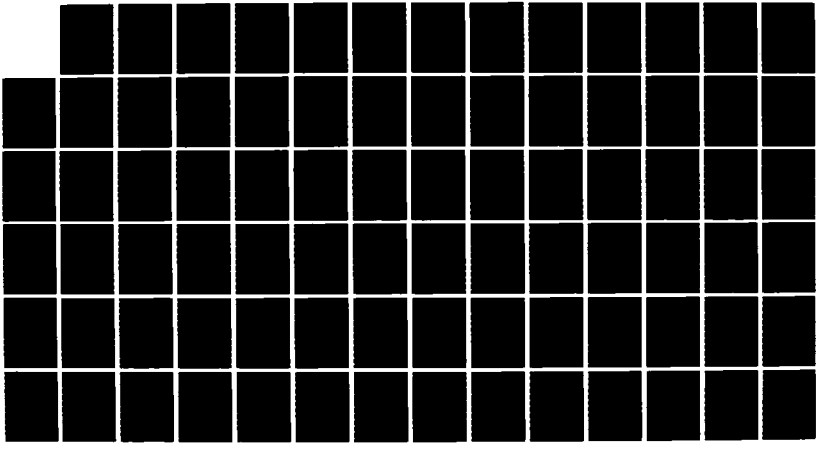
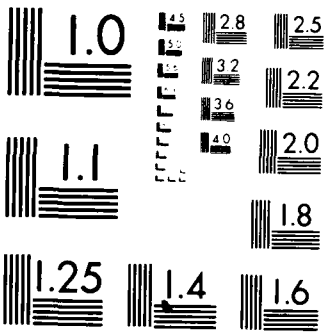
AD-A170 748 SHORT-TERM FILE REFERENCE PATTERNS IN A UNIX ENVIRONMENT(U) ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE R FLOYD MAR 86 TR-177 N00014-82-K-0193 1/1

UNCLASSIFIED F/G 9/2 NL





Short-Term File Reference Patterns in a UNIX Environment

Rick Floyd
Computer Science Department
The University of Rochester
Rochester, New York 14627

TR 177
March 1986

Department of Computer Science
University of Rochester
Rochester, New York 14627

Short-Term File Reference Patterns in a UNIX Environment

Rick Floyd
Computer Science Department
The University of Rochester
Rochester, New York 14627

TR 177
March 1986

Abstract

Data on short-term user file reference patterns have been collected from a local UNIX system supporting university research. These data provide detailed information on file opens and directory accesses. Somewhat coarser information on internal file operations has also been collected.

An analysis of the data shows that referenced files in our environment are generally small (under 1000 bytes), usually completely read or written, and have short interopen intervals (a succeeding open to a file usually occurs within 60 seconds of the last open). In addition, while there is extensive sharing of some files, this sharing is restricted to standard system files. We see very little sharing of user files.

This paper emphasizes the implications the patterns we observe have for distributed file systems. However, the results may also be applied to the design and modeling of more traditional file systems.

This work was supported in part by the National Science Foundation under grant number DCR-8320136 and in part by the Office of Naval Research under grant number N00014-82-K-0193.

1. Introduction

Much work has been done in recent years on transparent distributed file systems (DFS's) for local area networks [Ellis 83, Satyanarayanan 85, Tichy 84, Walker 83]. These DFS's typically provide transparent name lookup, transparent file access, and facilities for automatic caching and migration of files. Understanding and improving the behavior of such DFS's has been hampered by a lack of information on the ways that they are used. In particular, there is very little data available on short-term file and directory usage patterns.

Inspired and frustrated by this lack of information, we instrumented a local UNIX[1] system to collect information on file system requests. The UNIX file system is particularly appropriate for a study such as this one because it places relatively few constraints on user behavior. In addition, all of the DFS's mentioned above used the UNIX file system as their design model. We logged directory accesses, file opens, file closes, and information on process creation and destruction. In addition, the amount of information read and written for opened files was logged.

This paper describes the data collection method, presents our analysis of short term file reference patterns and discusses how the results may be applied to the design and tuning of DFS's. A companion paper [Floyd 86b] presents an analysis of short term directory reference patterns. We have generally tried to present results in a way that gives a qualitative feel for the characteristics of the data we have measured. Quantitative fits and distributions are, for the most part, sacrificed in favor of observations that would aid in developing and operating DFS's. These results, along with a simulation driven by the data we have collected, will be used to investigate the performance of the Roe distributed file system [Floyd 86a].

Our work is novel in several respects. It is by far the most detailed study of short term UNIX file reference patterns that has been done to date. It is also the only study we have seen that examines the differences between important user and file classes. In addition to examining the overall request behavior, we have broken down references by the type of file (temporary, log and permanent), owner of file (system, user and net[2]), and requester (system, user and net). We see large differences in behavior for the various classes. Knowledge of these differences should be useful in designing future DFS's.

Section two of this paper surveys previous work in the area. Section 3 describes the environment in which our measurements were made. Sections 4 and 5 present an overview of the data collection and analysis methods. In section 6, we present some of the results of the analysis. The implications the results of this analysis have for DFS design are discussed in section 7. Section 8 describes further analysis and data collection that could be done and section 9 summarizes our results. Finally, in the appendices, we present results that are too detailed for the main body of the paper.

Familiarity with UNIX [Ritchie 78] is assumed. Knowledge of 4.2BSD UNIX [Joy 83] may also be useful.

---

[1] UNIX is a trademark of AT&T Bell Laboratories.

[2] UUCP and USENET news (see section 5)

2. Previous Work

Early studies of file reference patterns [Satyanarayanan 81, Smith 81, Stritter 77] concentrated on long term (days or weeks) reference patterns that could be used in designing archival migration policies. The relatively small time delay of a local area network makes migration on a much smaller time scale feasible.

Recent work has concentrated on short term (seconds) file reference patterns. Porcar studied what was primarily batch activity on IBM mainframe systems [Porcar 82]. Our study is of an interactive UNIX system, an environment that has considerably different reference patterns. The differences between Porcar's results and ours are discussed in section 6.1. Satyanarayanan has made measurements on an interactive DEC-10 system [Satyanarayanan 83], with events being treated anonymously. Unfortunately, anonymous references make intelligent migration difficult. Smith has also done a study of short term file reference patterns [Smith 85], but primarily at the disk track level. We are concerned here with logical operations (opens, closes and so on) at a higher level of the system and less concerned with "tuning" using block caches and other such methods.

Work done on 4.2BSD UNIX file reference patterns at Berkeley is more closely related to our work. A file system tracing described by Zhou et al. [Zhou 85] is similar in approach to ours. The differences between the two packages are described in section 4. Ousterhout et al. report on read/write characteristics, file sizes, and data lifetimes for several 4.2BSD UNIX systems [Ousterhout 85]. Their results in these areas are similar to ours and are discussed in more detail in section 6.1.

None of these previous studies have collected information on directory access patterns. This information is not needed in systems that are concerned primarily with migration to manage disk storage, since files are typically much larger than the directories that reference them. However, a DFS may also migrate and replicate directories to improve performance and availability. In a DFS with non-trivial directory structures, the overhead of directory access is an important performance consideration. Evaluating directory design decisions in the absence of data on reference patterns is difficult. Section 4 of this paper describes the directory data collected by our tracing package.

Previous studies have also, for the most part, ignored the distinctions between batch and interactive use, system and user files, log and permanent files and so on. We believe that information on the behavior of each of these file classes can be of great value in designing a DFS and have considered them separately when clear differences exist.

3. Data Collection Environment

The data used in this paper were collected from a VAX 11/780 on the University of Rochester Computer Science Department Internet. At the time that the data was collected (September 1985), the internet consisted of a VAX[3] 11/780, 4 VAX 11/750's, 7 Sun workstations, 13 Xerox Dandelion workstations, 3 Symbolics LISP machines and a number of special purpose devices. The 11/780, Seneca[4], was selected as the primary machine for data collection because it was far and away the most heavily used of our systems. Seneca had, at the time, 4MB of memory, 560MB of disk storage and was running 4.2BSD UNIX. The

[3]VAX is a trademark of Digital Equipment Corporation.

[4]Our local VAXen are named after Western New York State's Finger Lakes.

system supported roughly 200 users. The primary user activities were program development (as part of our research effort), text editing and formatting, reading news and reading personal mail. Seneca also acted as a USENET news and UUCP mail relay [Nowitz 78]. There was relatively little database activity.

Data were also collected from two of the 11/750's. Preliminary analysis of the 11/750 data merely confirmed the importance of Seneca in our environment. Neither of the 11/750's had file system activity levels greater than 15% of that seen on Seneca. Because of this, only the Seneca data were fully analyzed.

4. Data Collection Method

Two types of data were collected: 1) a static "snapshot" of the file system and 2) a running log of file system activity.

4.1. Static Snapshot

The static snapshot provides a picture of the entire file structure on a machine at a given point in time. The information generated for each file system object that we are interested in is given in table 1. Processing starts at the root of the file system hierarchy and recursively traverses the directory tree, logging each object encountered.

A static snapshot was taken of the Seneca file system when file system logging (section 4.2) was started. This snapshot was used as a starting point for the analysis programs (section 5) and also provided information on the static file size distribution.

4.2. Logging File System Activity

The 4.2BSD UNIX kernel was modified to log selected system calls made by users. The calls logged can be classified as follows:

(1) Directory structure modifications: mkdir, rename, rmdir and symlink.

(2) Process context: chdir, chroot, exit, fork/vfork and setreuid.

(3) Other references: close, execv/execve, link, open/creat, truncate, unlink.

The logging of these calls has a negligible effect on the performance of the host (less than 1%).

object	output
directory	name, device, inode
regular file	name, device, inode, size (bytes)
symbolic link	name, target file
special file	name

Table 1: snapshot output

A number of other file-system related calls were judged unnecessary for our purposes (due to our ability to infer them from other calls or to their infrequent use) and were ignored. These included:

(1) Internal file operations: read, write, lseek. Actually, code was added to log reads, writes and seeks. However, running with this code enabled increased the size of log files by 500% and resulted in a 5-10% degradation in host performance. Since we were concerned primarily with operations on files as a whole, this additional overhead was felt to be unacceptable. Instead, we summarized some of the information in close records (see table 2).

(2) Protection: chmod, fchmod, chown, fchown.

(3) Status: readlink, fstat, lstat, stat, utimes, access.

(4) Other: fcntl, flock, fsync, mknod, ftruncate.

Each log record included the time that the call finished (with a resolution of 10ms) and the pid (process identifier) of the process making the request. In addition, most records contained information describing the call arguments and result. The record contents are given in table 2.

A brief explanation of the contents of table 2 is in order at this point. The first four records (mkdir, rename, rmdir and symlink), combined with the results of a static snapshot taken at the start of logging, allow us to construct and maintain a model of the directory tree for the file systems on the machine. *Mkdir* creates a new directory. *Rename* changes the path used to reach an object. *Rmdir* deletes a directory. *Symlink* creates a symbolic link containing a path to a file or directory. When a symbolic link is encountered during path resolution, the path in the symbolic link is substituted into the partially resolved path before resolution is continued. This is the only way in 4.2BSD UNIX to make links across file systems.

The next 5 records (chdir, chroot, exit, fork and setreuid) give us the information we need to keep track of the working directory and real uid (ruid) of each process. *Chdir* changes the directory used to resolve

call	output
<all>	time, pid of caller
mkdir	+ file id of new directory, path of new directory
rename	+ old path, new path
rmdir	+ path of deleted directory
symlink	+ target of link, link name
chdir	+ path to new working directory
chroot	+ path of new root
exit	-
fork (fork/vfork)	+ child pid
setreuid	+ new ruid
close	+ file id, final size, bytes read, bytes written
execute (execv/execve)	+ file id, uid of file owner, size, path
link	+ target path, link path
open (open/creat)	+ file id, open flags, mode of file, size, uid of file owner, path
truncate	+ path, new size
unlink	+ path

Table 2: dynamic log structure

relative references made by a process (those not starting from the root of the file system tree). *Chroot* changes the root of the file system as seen by a process. *Fork* (fork and vfork system calls) and *exit* create and destroy processes. Logging these allows us to keep track of processes created for each user. *Setreuid* changes the effective "owner" of the current process. This is the mechanism for logging into the system.

The remaining records (close, execute, link, open, truncate and unlink) are the actual references to files. *Execute* (execv and execve system calls) executes a file, replacing the current process with the image given in the file. *Link* and *unlink* add and delete directory entries for files. If unlink removes the last link to a file, the file is deleted. *Open* (open and creat system calls) opens or creates a file or opens a directory. Processes access files either by explicitly opening them or by inheriting open files from their parents. *Truncate* shortens a file. *Close* records indicate that a process no longer has a file open. They are generated by either a close system call or by a process exit. As mentioned earlier, a process may inherit open files from its parent. If this happens, the close record is generated when the last process having access to a file due to the open closes the file or exits (for those in the know: we log the release of the kernel open file table entry). We only log closes for regular (data) files. Closes are not logged for directories or for special files (files corresponding to devices). Since directories are short, completely scanned when opened and can only be opened for reading, close records for directory opens would have given us little useful information. Special files are not analyzed in this study (except for a count of opens).

The calls listed in table 2 are logged for all processes in the system. In addition, a small number of administrative records having to do with enabling and disabling data collection are logged. The most important of these is the *process state* record. A process state record contains information on the ruid, working directory, root directory and command name for a process. One of these is logged for a process the first time it appears in a log, but only if we don't already have this information for the process. Process state records are only necessary for processes that exist before logging is started (and for their children until we log the parent). They give us a way to locate the process in the directory tree and to classify it as a user, system or net process.

The 4.2BSD tracing package we have described differs from the one developed independently at Berkeley by Zhou et al. [Zhou 85] in a number of ways. The most important difference is that we don't collect information on internal file operations. This means that we have less information on the timing of these operations to files and on which bytes are accessed. We do, however, log the number of bytes read from or written to an opened file. As we will see later on (section 6.1), most files in our environment are read or written completely and are usually open for only a short period of time. These results, combined with the fact that most DFS's treat files as a whole, mean that the omission of internal file operations is not important for our particular application.

We also collect less information per record. In particular, all of our times are real times at the finish of the system call. Zhou et al. record, in addition to real times, the duration of the call and process virtual times. We made a decision early on to collect the minimum information necessary for our purposes. This allows us to collect and process data for a longer period, but means that our trace is sensitive to the capacity of the machine that the data was collected on. Adjusting for this would be difficult in any case.

Finally, we collect information on high level directory operations (create, delete and open). This allows us to track process locations in the directory tree so that we can accurately analyze relative file references. It also gives us the data needed to analyze directory reference patterns.

The trace data collected by Ousterhout et al. [Ousterhout 85] includes information on seeks (so that read and write data may be derived), but lacks information we record that allows references to be classified by file type and file owner. We also include directory and process information not present in their trace.

Note that our package does *not* collect a full trace of file system activity. We don't collect information on inode accesses, paging activity, internal file operations (except for the total number of bytes read and written), or protection and status related calls. However, our package does generate detailed information on the most common operations on files and directories as a whole (open, close, create, delete, execute and so on) and on overall read and write activity for opened files. This information provides a useful basis for investigating file usage patterns and is sufficient for trace driven studies of most DFS's.

5. Analysis Method

5.1. Basic Approach

The data in the raw form described in table 2 is difficult to analyze. There is no obvious correspondence between opens and closes, unlinks are not associated with the files they affect, no direct information is available on process working directory or owner and so on. A library of analysis routines was written to address these difficulties. The routines maintain enough state about the system being analyzed to allow the necessary associations to be made. Alternatives would have been to reformat the file reference logs so that each record contained the information necessary for its analysis (see, for example [Zhou 85]) or to collect more information for each reference. We chose to derive the information at the time of analysis to minimize the disk resources needed (and so maximize the logging period). Of course, one pays a penalty in analysis time for doing this. Using this approach, a simple analysis of the trace described in this paper (2.5 million events occupying 70MB of disk) takes about 5 hours of 11/780 CPU time. This is adequately fast for our needs.

Analysis proceeds in two phases. During the initialization phase, a snapshot of the directories in the system being analyzed is read in and used to set up a model of the original directory structure. During data analysis, log records are read and passed, one by one, to user analysis routines. These log records are also used to update state information on files, directories and processes in the system, creating and destroying them to maintain an accurate model. Given this up to date state information, the library routines can perform the associations mentioned above and pass this information on to the user routines.

There are some conventions worth mentioning here that are used by all analysis programs:

(1) Calculations involving file sizes are always based on the size of the file when it is closed or executed.

(2) File reads and writes are assumed to occur at the time a file is closed (we didn't have more accurate information on these operations). Since the time most files are open is usually considerably shorter than any of our histogram resolutions, this has no noticeable effect on our results.

(3) File lifetimes run from the time a file is created (based on a create flag in the open call) until the time the underlying inode supporting the file is deleted. This doesn't happen until there are no links to the file left *and* there are no active opens, so the delete time can (and frequently does) differ from the time of the last unlink. File version lifetimes are handled in a similar

fashion.

(4) Processes occasionally open a file and then open it again before closing it. This is usually done to get both read and write access to a file without using the mechanisms for this built into the 4.2BSD kernel. We honor the intent (not the method) by combining these opens into one open with both access modes. This affects only 0.7% of the opens and so is not an important consideration in any case.

5.2. Cuts

We are interested in investigating both the overall pattern of requests to the file system and in the patterns for various classes of users and files. Past work has often ignored the distinction between batch and interactive use, system and user files, log and permanent files and so on. We believe that information on the behavior of each of these file and user classes can be of great value in developing a DFS and have developed a number of data cuts to separate the classes of interest. We use three basic types of cuts:

(1) Cuts on the ruid (owner) of processes making requests (UUCP/USENET network, system and user).

(2) Cuts on the owner of files (UUCP/USENET network, system and user).

(3) Cuts based on the purpose of files (log, permanent, temporary).

Some of these can be combined to give other more specific cuts. 14 cuts are used in this paper. The cuts and their meanings are:

(1) **no cut**: This cut passes all records in the log to the user analysis routines.

(2) **ruid_NET**: Passes references by what we term *net* processes. Net processes are those running under UUCP, USENET news or notes accounts. Most of these processes run in batch mode and so this cut gives us a sample that is considerably different from an interactive one. This category has been broken out from the system and user categories because of the batch-oriented nature of the references and the large number of references by net processes (roughly 1/3 of the references in this study and as much as 70% of the non-system references in earlier studies [Floyd 85]). We don't include references due to Seneca being on the Rochester Internet in the ruid_NET category.

(3) **ruid_SYSTEM**: Passes references by system processes (those running under root, daemon, games and other miscellaneous system accounts). System processes are primarily daemons that provided widely used services (such as spooling and network status reporting), processes created on behalf of users to perform privileged operations, and periodic maintenance processes.

(4) **ruid_USER**: Passes references by processes running under user accounts.

(5) **owner_NET**: Passes references to files owned by UUCP, USENET news and notes accounts. These are primarily news articles and UUCP spool files.

(6) **owner_SYSTEM**: Passes references to files owned by the system accounts mentioned above. This includes major administrative and status files (for example, /etc/passwd), system libraries, system include files and so on.

(7) **owner_USER**: Passes references to user files.

(8) **file_LOG**: A number of files on any UNIX system are used to keep logs of activity. Examples include /usr/adm/messages, /usr/adm/wtmp and user mbox files. Since we expect the access patterns for these files to be considerably different from that for files as whole and since these files are generally quite large, we use a cut, file_LOG, that allows us to analyze only these logs.

We had originally intended to place in this category just those files opened with append-only access. However, it soon became clear that this mode of access is basically never used. Instead, most logs are opened write-only, a seek is done to the end of the file and then the log entry is appended. If several processes are trying to update a log simultaneously, the results are unpredictable. Some of the busier logs on our system are scrambled on a regular basis using this "method."

We were eventually forced to use the name of the file given in the open call to make this cut. Luckily, most of the log files on the system have well known names and an examination of source for commonly run programs and of the file reference logs enabled us to find the rest of the log files on the system.

(9) **file_PERM**: Passes references to permanent files. This includes all files that aren't log files (file_LOG) or temporary files (file_TEMP).

(10) **file_TEMP**: Passes references to temporary files. This includes files that are created on a special file system (/tmp), temporary spool files, lock files and other such transitory files. Most temp files are clearly identified by either their name (a special template is usually used to create temp file names) or by the directory in which they are created.

(11) **owner_USER + ruid_USER** (shown as U in tables and figures): Passes references that satisfy both the owner_USER and ruid_USER cuts. These are user references to user files. The owner_USER + ruid_USER cut produces results similar to the owner_USER cut. There are about 9.5% fewer references for the U cut, but the resultant distributions are nearly identical. It is included here for comparison with the next three cuts.

(12) **owner_USER + ruid_USER + file_LOG** (shown as U + file_LOG in tables and figures): Passes user references to user log files.

(13) **owner_USER + ruid_USER + file_PERM** (shown as U + file_PERM in tables and figures): Passes user references to user permanent files.

(14) **owner_USER + ruid_USER + file_TEMP** (shown as U + file_TEMP in tables and figures): Passes user references to user temporary files.

5.3. Analysis Complications

The data analysis did not proceed as smoothly as we had hoped. This section describes some of the problems we experienced and suggests changes in the data collection and analysis that would help avoid these problems in the future. None of these problems was serious enough to have a noticeable effect on our results.

It was not always possible to pair up opens and closes correctly. In most cases there was only one open for a given file to associate a close with or the process numbers of an open and close matched. In cases were this was not true, we looked for an open that was made by an ancestor of the process making the close request. Sometimes there were multiple opens to a file outstanding among those made by ancestors. This problem occurred less than 0.03% of the time and was dealt with by using the most recent open by an

ancestor. A more accurate solution would have been to record an open session number in the log and use this to make the association, but the low frequency of occurrence of this problem and the relative unimportance of the derived numbers makes this solution unnecessary for us.

There were two peculiarities in the 4.2BSD kernel that resulted in some surprises in the logs. One, having to do with incorrect returns from the fork call, was caught and corrected before the data analyzed here was collected. The other, an inconsistent handling of error indicators when a process was forcibly terminated, caused us to lose some close and exit records. This was not discovered until fairly late in the analysis. Less than 0.03% of the close records and about 0.5% of the exit records were not recorded because of this problem. Since the number of close records lost was so low, we made no attempt to correct the problem.

Files were classified (as log, perm or temp) the first time they were seen in the logs. Occasionally this classification was incorrect. While we were developing the cuts, we classified a number of files by hand, using information on the programs making the requests and the full history of reference patterns to files. Comparing our classifications to those done by the analysis routines (using file name and directory information) showed that only a few tenths of a percent of files were incorrectly classified. It would be difficult to do better than this without explicit information on the intended usage of files. This information is just not available under UNIX.

We had to retain a large amount of state in order to associate unlink records with files and to interpret their meaning. Since we needed most of this state for other reasons (uid classification, directory studies and so on) this was not really a problem for us. Including the file id and a count of the number of remaining references in unlink records would make it possible to interpret them in the absence of the state information.

6. File Reference Patterns

Roughly 7 days of data were collected on Seneca (168.82 hours, from 3:21am on Monday, September 16, 1985 to 4:10am on Monday, September 23). During this period there were 142 active users of the system. There were generally 20 to 30 logged in users at any given time on weekday afternoons, with load averages running between 5 and 10.

In section 6.1, we examine the overall pattern of open and read/write requests. Section 6.2 briefly examines execve patterns. Section 6.3 concentrates on user files. Our approach in all cases is to present only those tables and histograms that are particularly characteristic or striking. A more complete breakdown of many of our results may be found in the appendices.

6.1. Overall Open and Read/Write Patterns

6.1.1. Basic Statistics

A summary of the records collected is given in table 3. The first three columns give the number of records of each type collected, the average rate for that type of record, and the percentage of the collected records that this represents. The remaining columns show the number of records collected cut by the ruid of the calling process and the percentage of the total for the ruid class.

From this table we can see that each of our ruid categories accounted for roughly 1/3 of the activity on the system. The majority of the file system requests were for opens and closes, with most of the rest of the categories being a factor of 5 or more down from this (of course we didn't record reads, writes and seeks, all of which would be a significant component of a full trace). Processes made, on the average, 5.3 open requests.

Table 4 gives the number of opens to each type of file object on the system. For the purposes of comparison, the SLAC trace [Porcar 82] included about 237,000 opens to data (regular) files in a similar period. The remainder of the analysis in this paper deals with only regular files, the largest category in table 4. Directory access patterns (including explicit directory opens) are analyzed in a companion paper [Floyd 86b]. Block and character special files are used in UNIX to provide access to devices and are not of interest to us. They are, in any case, a small fraction of the total number of opens.

record	no cut			ruid_NET		ruid_SYSTEM		ruid_USER	
	count	per hr	fraction	count	fraction	count	fraction	count	fraction
mkdir	936	5.5	0.04%	795	0.11%	2	0%	139	0.02%
rename	3211	19	0.13%	1946	0.26%	408	0.04%	857	0.11%
rmdir	913	5.4	0.04%	780	0.11%	0	-	133	0.02%
symlink	16	0.1	0%	0	-	3	0%	13	0%
chdir	136063	806	5.4%	19102	2.6%	71854	7.1%	45106	5.7%
chroot	0	-	-	0	-	0	-	0	-
exit	180270	1070	7.1%	31219	4.2%	85917	8.5%	63133	8.0%
fork	181511	1080	7.1%	29271	4.0%	90735	8.9%	61503	7.8%
setreuid	16772	99	0.66%	4372	0.59%	9698	0.95%	2701	0.34%
close	754072	4470	29.7%	249837	34.0%	298164	29.4%	205666	26.2%
execute	125064	741	4.9%	26761	3.6%	38093	3.8%	60209	7.7%
link	42929	254	1.7%	25694	3.5%	7301	0.72%	9934	1.3%
open	965087	5720	38.0%	277350	37.7%	393661	38.8%	294070	37.4%
truncate	0	-	-	0	-	0	-	0	-
unlink	130929	776	5.2%	68342	9.3%	19861	2.0%	42726	5.4%
total	2537773	15040	100%	735469	100%	1015697	100%	786190	100%

Table 3: records logged

type	no cut		ruid_NET		ruid_SYSTEM		ruid_USER	
	opens	fraction	opens	fraction	opens	fraction	opens	fraction
regular file	754285	78.2%	249825	90.1%	298186	75.7%	206268	70.1%
directory	170448	17.7%	17275	6.2%	72625	18.4%	80548	27.4%
block special	922	0.1%	0	-	60	0.02%	862	0.3%
character special	39432	4.1%	10250	3.7%	22790	5.8%	6392	2.2%
total	965087	100%	277350	100%	393661	100%	294070	100%

Table 4: Opens, by object type

Opens may be further broken down by the class of file being opened and by the owner of the file. This information, plus statistics on how many files there are in each category, is given in table 5. We see here that 2/3 of the references were to perm files, although temp files made up 4/5 of the files referenced. Relatively few references were made to user files. The large number of net files may be attributed to a daily news expiration procedure that reads the headers of all news articles.

Information on read/write modes for open-close sessions is given in table 6 (note that percentages in this table sum horizontally). Overall, files opens were evenly split between opens with read-only access and opens for write-only or read-write. Users, however, opened most files read-only. Log files were generally opened write-only.

Perm files are categorized by their function in table 7. This categorization was done using the directories that files appeared in and/or based on file names and extensions. "System configuration" files are those appearing in / and /etc. Examples are /vmunix (the bootable kernel image) and /etc/passwd (passwords and other information on accounts). "Rwho daemon" files are used to maintain status information about machines on the network. "Library" files are those in /lib, /usr/lib and so on (this includes both program libraries and additional configuration files). Files with names beginning with '.' are grouped into the category "personal configuration." These files traditionally contain startup commands and status information for various programs and are used to tailor and maintain an individual's environment.

cut	opens	% opens	files	% files	opens/file
file_LOG	35662	4.7%	506	0.5%	70.5
file_PERM	499193	66.2%	16352	16.2%	30.5
file_TEMP	219430	29.1%	84327	83.3%	2.6
owner_NET	249733	33.1%	46207	45.7%	5.4
owner_SYSTEM	392790	52.1%	25062	24.8%	15.7
owner_USER	111762	14.8%	30822	30.5%	3.6
no cut	754285	100%	101185	100%	7.5

Table 5: Class and owner of opened regular files

cut	read-only		write-only		read/write		total
	opens	fraction	opens	fraction	opens	fraction	opens
file_LOG	735	2.1%	34819	97.7%	97	0.3%	35651
file_PERM	282853	56.7%	180976	36.3%	35200	7.1%	499029
file_TEMP	104828	47.8%	96766	44.1%	17794	8.1%	219388
owner_NET	148150	59.3%	79739	31.9%	21830	8.7%	249719
owner_SYSTEM	175787	44.8%	198183	50.5%	18712	4.8%	392682
owner_USER	64479	57.7%	34639	31.0%	12549	11.2%	111667
ruid_NET	146993	58.8%	79111	31.7%	23713	9.5%	249817
ruid_SYSTEM	99205	33.3%	188233	63.1%	10723	3.6%	298161
ruid_USER	141822	69.0%	45189	22.0%	18654	9.1%	205665
no cut	388416	51.5%	312561	41.4%	53091	7.0%	754068

Table 6: Mode of open for open-close sessions

Examples include .login, .profile and .newsrc. The rest of the categories have the obvious meaning. Note that over half of the opens to perm files were made to 0.7% of the files (those in the first two categories). These files were basically all system configuration and status files. Activity to these two categories represents roughly 40% of the total file opens we observed, indicating that a substantial fraction of the system activity was devoted to communicating and maintaining information about itself and about other hosts on the network.

6.1.2. Per Open Results

The open activity over time is shown in figure 1. Opens followed a daily pattern with a busy period between 9am and 6pm, overlaid by strong bursts due to net activity (mostly news expiration and news reception). Weekends were relatively quiet.

Figure 2 plots the open activity for just the first day of the trace. This shows the work day busy period more clearly. Looking closely, we can see that user activity accounted for roughly half of the daytime load. System opens had a base level (the rwho daemon) overlaid by activity that followed or lagged slightly behind user and net activity. That is, a significant part of the system activity was indirectly due to the other classes. This activity may be attributed to logins, spoolers, mailers and so on.

The read and write activity to regular files corresponded only roughly to the open activity. This can be seen by comparing figures 3 and 4 with figure 1[5]. Reads and (especially) writes were fairly bursty on the resolution used in these figures (about 2 hours). The burstiness increased as the resolution used increased. Figure 5 shows the throughput of the file system during a typical period of heavy user activity, averaged over 10 second intervals. This represents activity for about 25 logged in users. It is interesting to note that the peak rates in this figure, 35K bytes/second, would present little problem for today's LAN technologies, even with fairly hefty open and transfer protocol overheads. Our results here are similar to those presented by Ousterhout et al. [Ousterhout 85] and supports their contention that such networks can support large numbers of users.

category	opens	% opens	files	% files	opens/file
system configuration	123481	24.7%	100	0.6%	1235
rwho daemon	166761	33.4%	13	0.1%	12830
library	59245	11.9%	222	1.4%	267
manual pages	18371	3.7%	1597	9.8%	11.5
news	40022	8.0%	5828	35.6%	6.9
program source	10596	2.1%	1499	9.2%	7.1
includes	13767	2.8%	344	2.1%	40
objects	5618	1.1%	468	2.9%	12
personal configuration	23125	4.6%	1676	10.2%	13.8
mail spool	3621	0.7%	524	3.2%	6.9
other	34586	6.9%	4081	25.0%	8.5

Table 7: Function of opened perm files

[5]The unusually heavy read/write activity on Thursday was caused by repeated execution of a large user text formatting job (formatting a Ph.D dissertation). Most of the activity was to temp files.

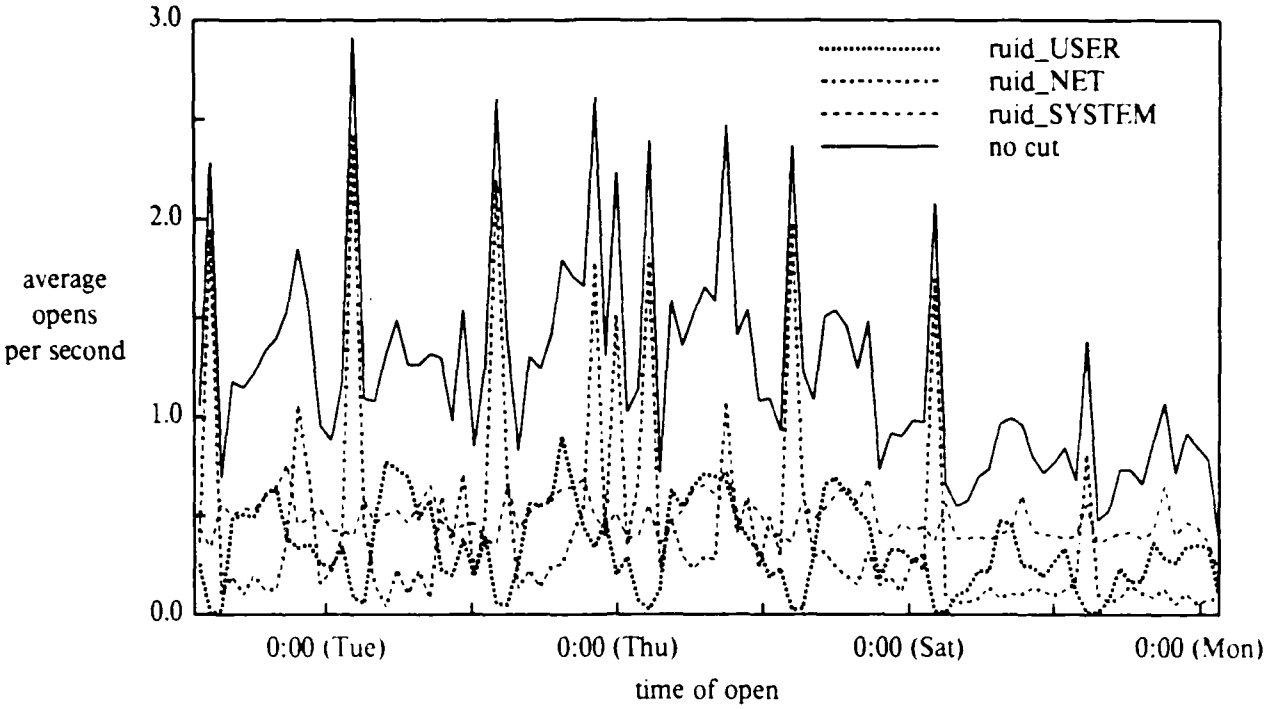


Figure 1: Average number of regular file opens per second (~2 hour resolution)

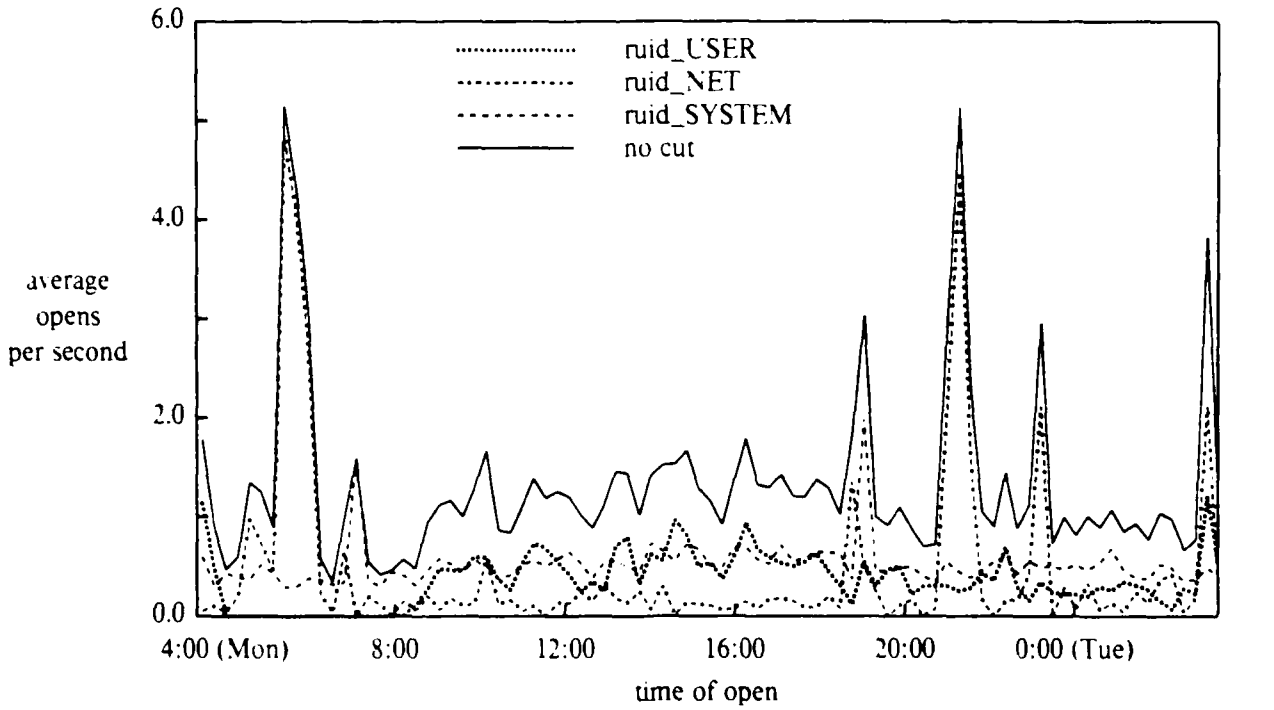


Figure 2: Average number of regular file opens per second (~15 minute resolution)

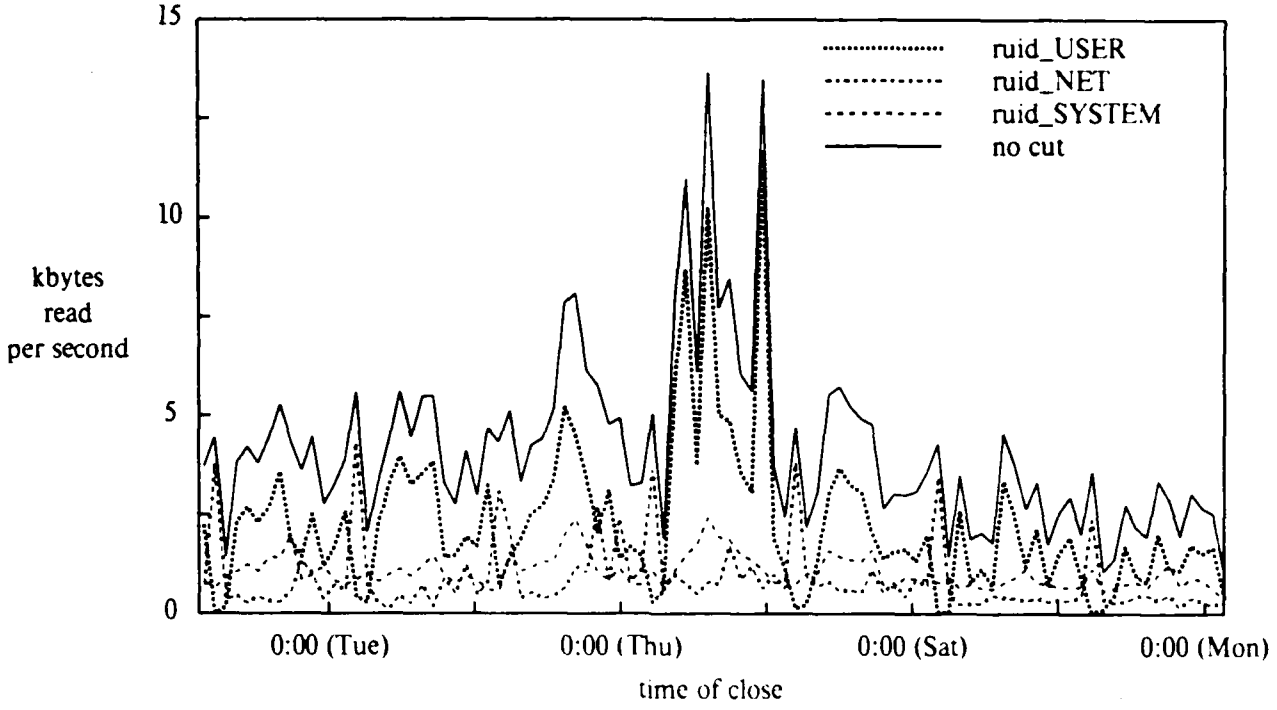


Figure 3: Bytes read from regular files (~2 hour resolution)

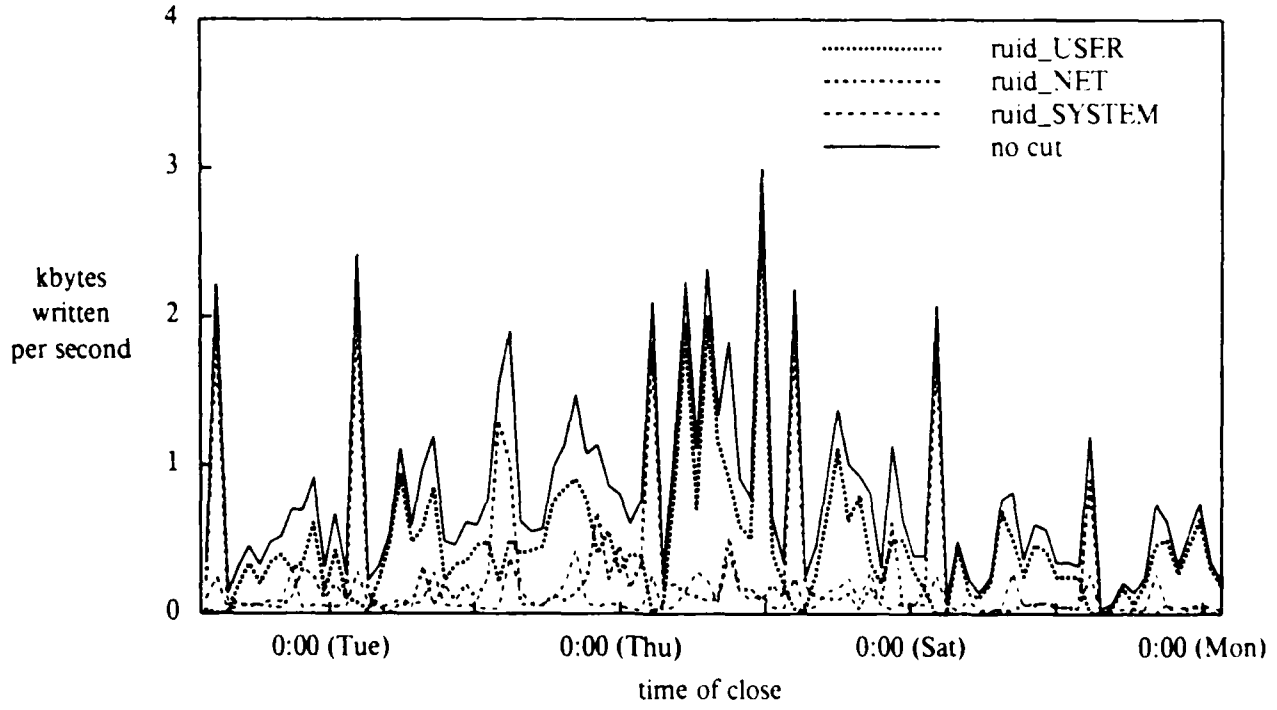


Figure 4: Bytes written to regular files (~2 hour resolution)

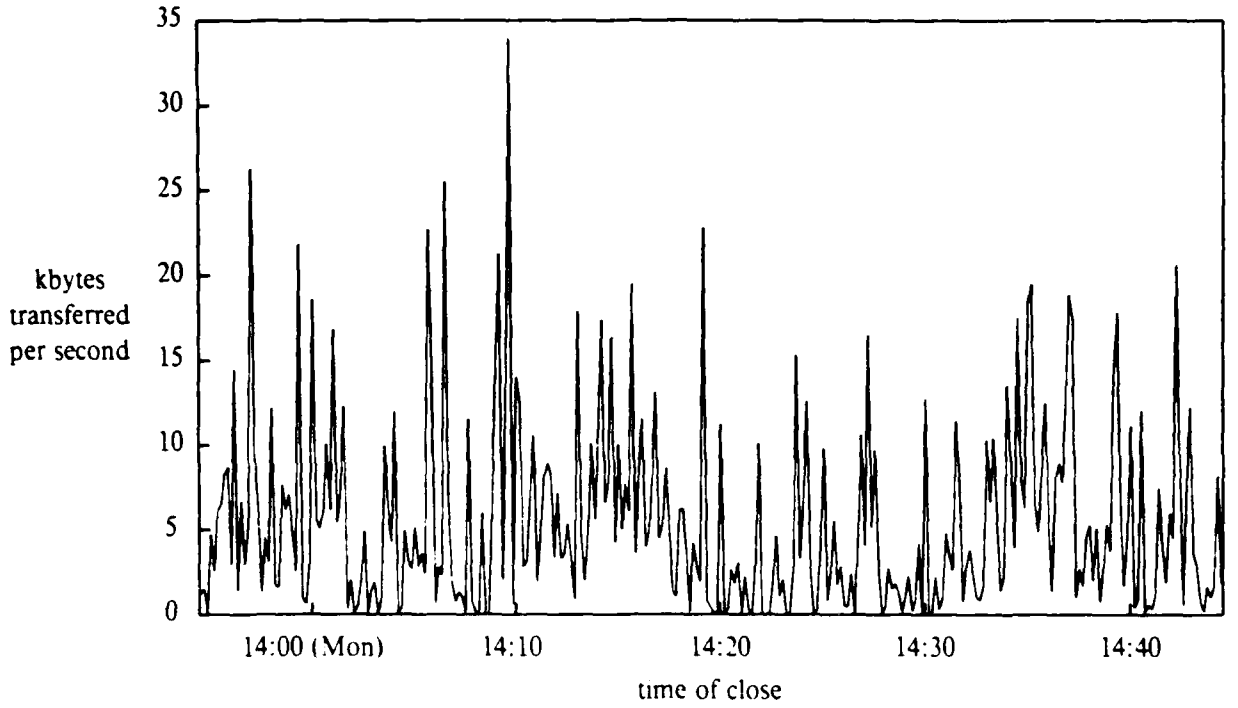


Figure 5: Bytes transferred to and from regular files (10 second resolution)

Table 8 shows the average throughput of the file system over the life of the trace for each class of user. Note that reads accounted for 84% of the bytes transferred. Users accounted for over half of all bytes transferred, even though they made only about a quarter of the opens to regular files (table 4).

Referenced files on Seneca tended to be small, particularly compared to IBM mainframe environments such as the ones studied by Porcar. Figure 6 and table 9 show file size distributions on Seneca, weighted by the number of opens made and cut by the class of file. Note that these are *cumulative* distributions. At any point on a curve, the y value is the fraction of files with sizes less than or equal to the x value. For comparison purposes, we have included here the static file size distribution, as derived from a snapshot of the file system taken at the beginning of data collection (this is the distribution that would result if each file on the system were opened once). Table 9 also includes statistics for on-disk permanent files referenced during the SLAC trace.

cut	reads		writes		overall (r+w)	
	bytes/sec	fraction	bytes/sec	fraction	bytes/sec	fraction
ruid_NET	870	21%	250	31%	1120	22.5%
ruid_SYSTEM	1060	25%	110	14%	1170	23.5%
ruid_USER	2260	54%	440	55%	2700	54%
no cut	4190	100%	800	100%	4990	100%

Table 8: Bytes read/written for regular files

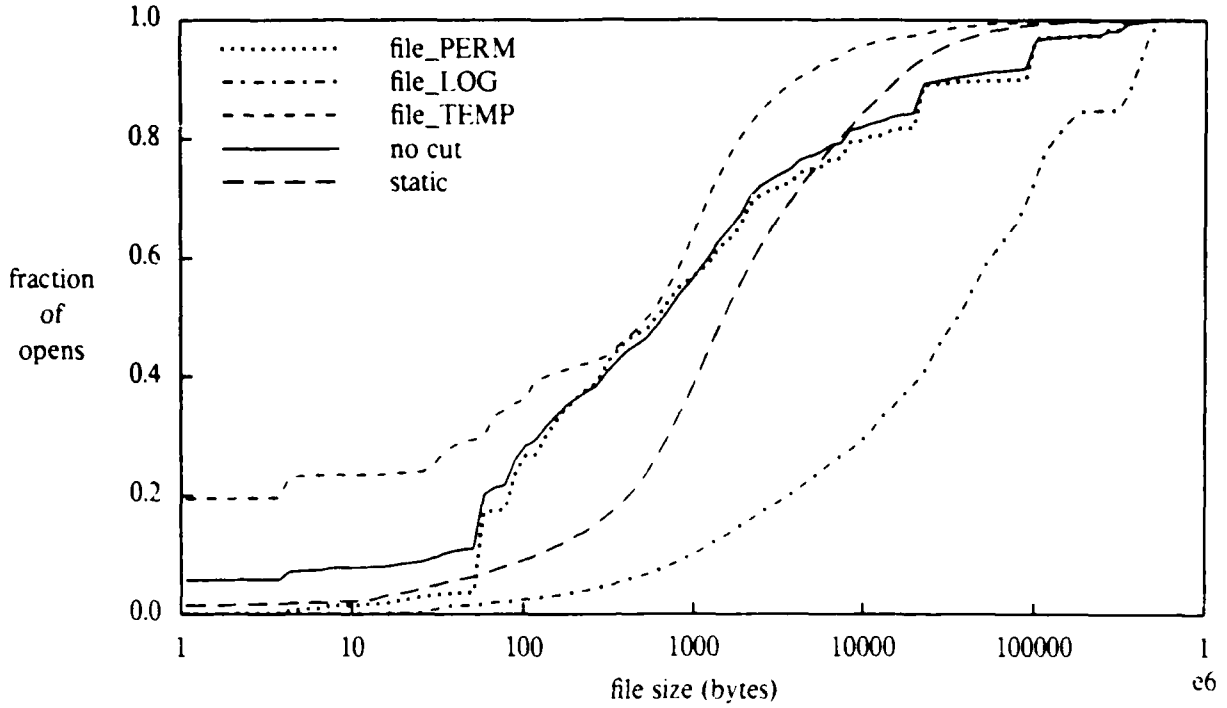


Figure 6: Dynamic file size distributions (cumulative, measured at close)

distribution	min	max	mean	median	std deviation
file_LOG, dynamic	0	1.28e6	105000	38900	1.5e5
file_PERM, dynamic	0	2.49e6	19600	620	5.9e4
file_TEMP, dynamic	0	1.3e6	2980	620	1.9e4
all, dynamic	0	2.49e6	18800	710	6.2e4
all, static	0	7.95e6	8020	1600	5.6e4
SLAC, disk file_PERM	0	94.0e6	549000	80000	2.3e6

Table 9: File size distributions

From figure 6 we see that there were substantial size differences between opened log, perm and temp files. The large number of zero length temp files was due to frequent creation of lock files (these lock files serve as a very crude mutual exclusion mechanism). Log files, on the other hand, were generally an order of magnitude or more bigger than other files. The jump at 60 to 100 bytes in the perm file distribution was due to the rwho daemon, which was updating a set of status files describing machines in the network every 60 seconds. By comparing the dynamic and static distributions, we find that opens tended to favor small files (due to lock and rwho daemon files) and, to a lesser extent, a few larger files (administrative files such as /etc/passwd).

The small size of opened files (55% are under 1024 bytes, a common block transfer size, and 75% are under 4096 bytes) suggests that directory lookup and open overhead will play a large part in file access times, particularly in a distributed environment.

While most files opened in our environment were small, the majority of bytes came from files that were much larger; 2/3 of all bytes were read from files greater than 20,000 bytes long. This is shown by figure 7 and table 10, which give distributions for the size of opened files, weighted by the number of bytes read. We have also included here, for comparison purposes, the static space used distribution (the distribution that would result if each file on the system were completely read once). The staircase effect in the dynamic distributions is due to repeated opens and reads of a few large administrative files. /etc/passwd, for example, at 21,000 bytes, accounts for almost 20% of the bytes read. This file is infrequently modified and so would be a good candidate for replication in a distributed environment. We saw earlier (table 7) that a relatively small number of files received a high fraction of the open traffic. Figure 7 gives graphic evidence of the corresponding impact on I/O traffic.

Our distributions for the overall sizes of opened files and for the source of bytes read (figures 6 and 7) agree with the distributions found by Ousterhout et al.. By these measures, at least, our data appears to be

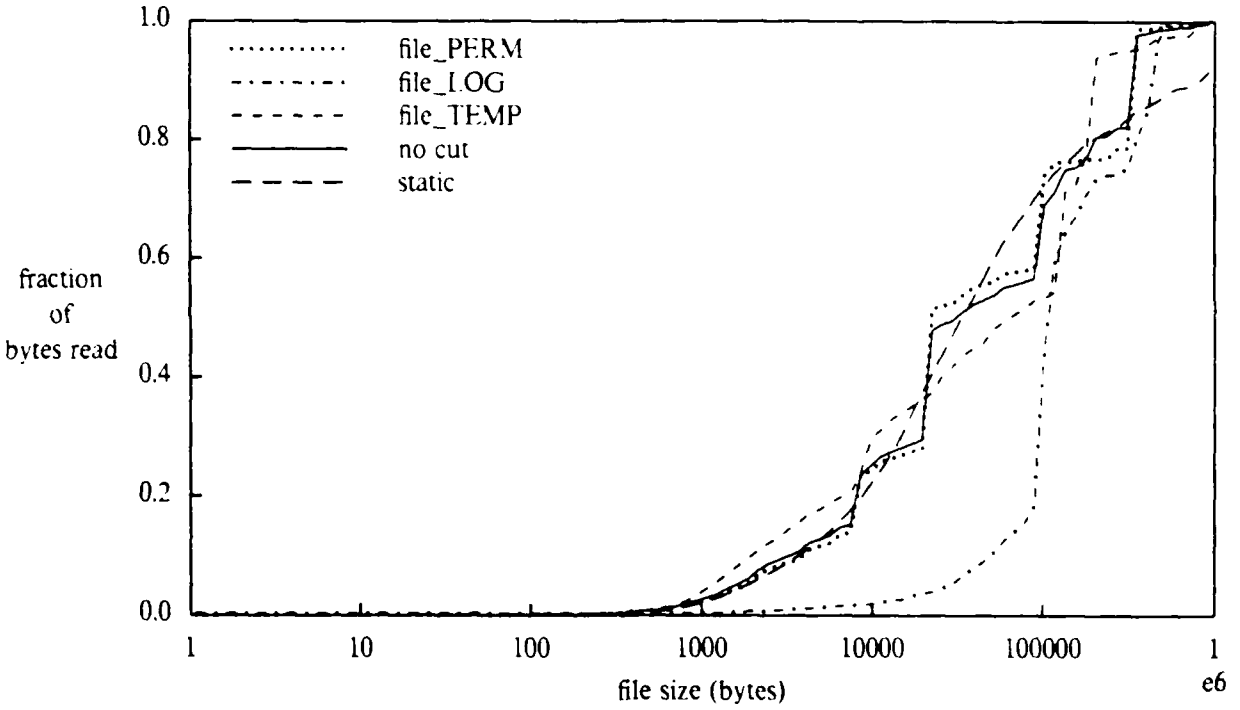


Figure 7: Dynamic file size distributions, weighted by bytes read (cumulative, measured at close)

distribution	min	max	mean	median	std deviation
file_LOG, dynamic	0	1.28e6	1.91e5	1.2e5	1.6e5
file_PERM, dynamic	0	2.49e6	1.19e5	2.2e4	2.0e5
file_TEMP, dynamic	0	1.3e6	1.12e5	6.8e4	1.5e5
all, dynamic	0	2.49e6	1.19e5	3.4e4	1.9e5
all, static	0	7.95e6	3.6e4	1.29e4	5.1e4

Table 10: File sizes, weighted by number of bytes read

representative of a university research environment.

Two figures that are useful in estimating the appropriateness of dynamic migration are the fraction of a file opened for reading that is actually read and the fraction of a file opened for writing that is actually written. As mentioned in section 4, we don't have complete information on which bytes of opened files were read and written. However, if we make the reasonable (for our environment) assumption that a given byte in a file was not usually read or written repeatedly in a single session, we can use the counts of bytes read and written from the close record to calculate the fraction of a file read or written. Figure 8 shows the percentage read for files opened read-only, cut by the class of the file. Figure 9 shows the percentage written for files opened write-only and figures 10 and 11 are for files opened read/write. In all cases, the size used is the size of the file when closed. Zero length files are omitted. Tables 11-14 provide some statistics on the distributions in these figures.

From these figures we see that most opens with read-only or write-only access resulted in the file being completely read or written. The notable exception was for log files. For these files, writes usually just incrementally extended the file. This is shown clearly in figure 9 and indicates that we have successfully extracted log files from our data. Much less can be said about the read/write behavior of files opened with read/write access. For these files, information on usage history or more detailed information on the intended usage of the file would be needed to predict the read/write behavior. Recall (table 6) that this category represents only 7% of the opens and so the additional information will not usually be needed.

Overall, 68% of files opened with read access (read-only or read/write) were completely read and 78% of files opened with write access (write-only or read/write) were completely written. This may be contrasted with the SLAC data, where only 17% of opened permanent files were completely accessed. The high percentage of files completely accessed on Seneca is due to the much smaller file size and to the lack of any serious database activity.

As one might expect, the fraction of a file that was accessed depended strongly on the size of the file. Very small files were usually completely read or written. Large files were rarely completely read or written. This is shown for files opened read-only and write-only in figures 12 and 13 and in tables 15 and 16. Files opened with read/write access followed a similar pattern.

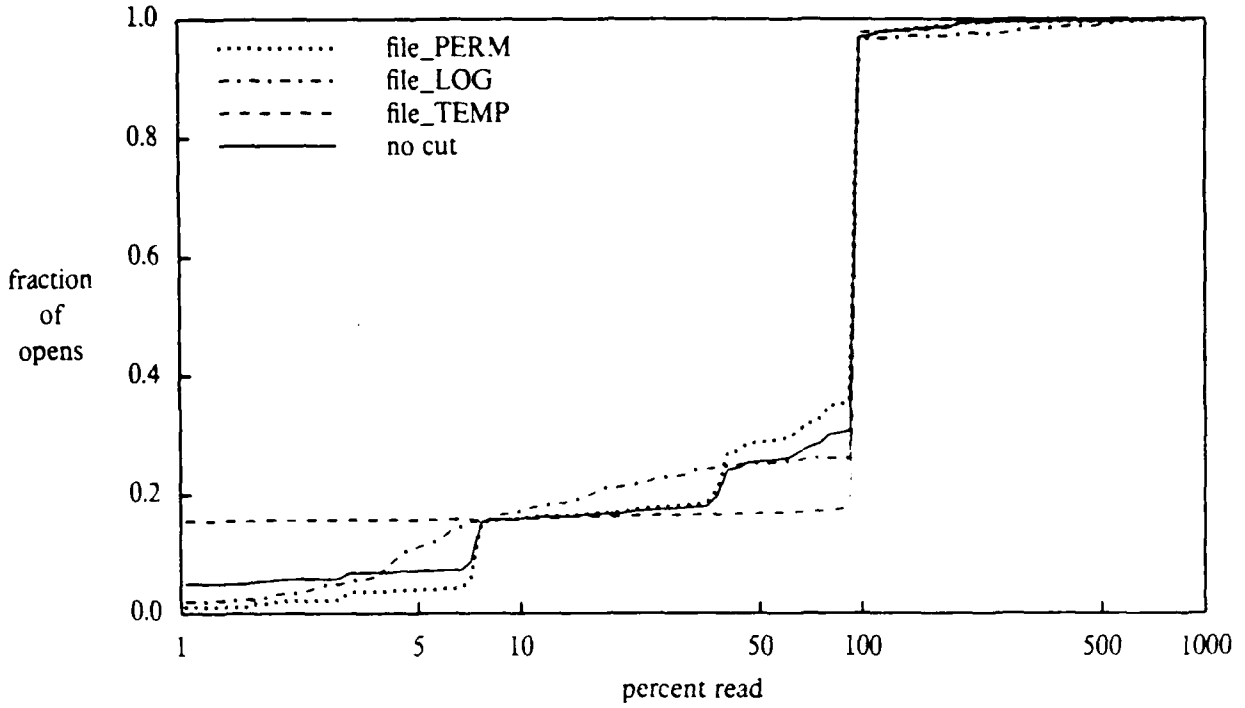


Figure 8: Percent of file read for read-only opens (cumulative)

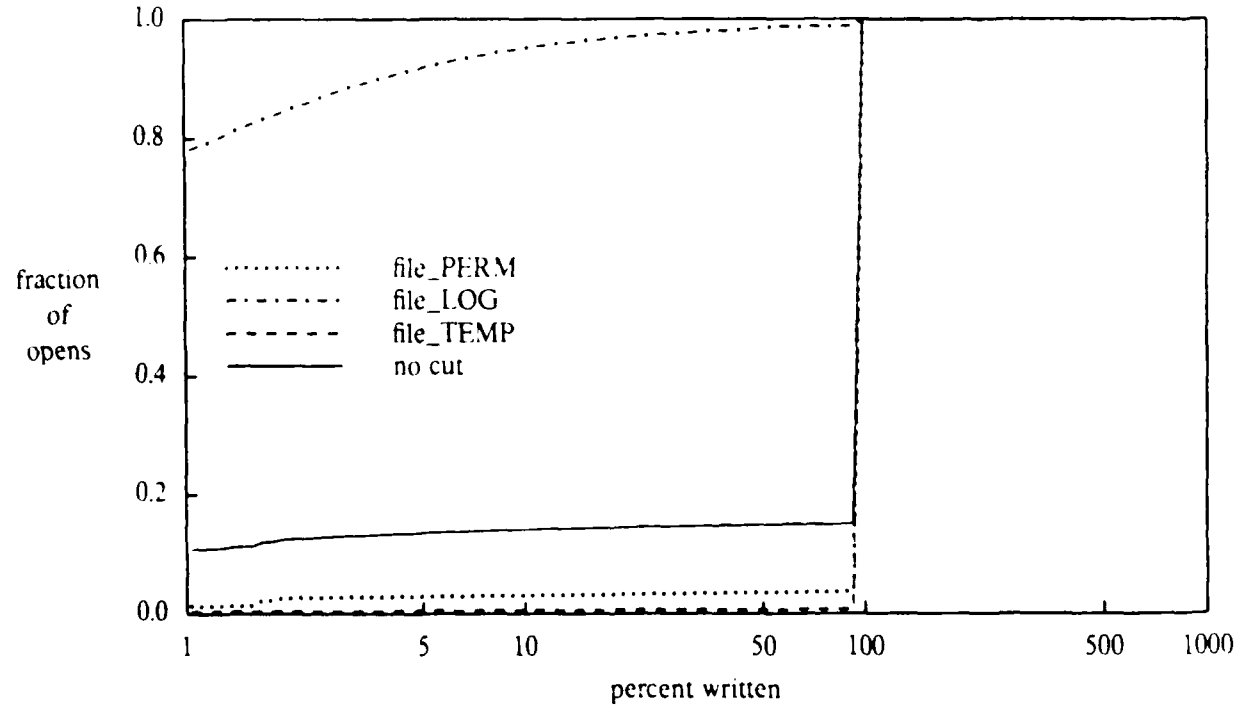


Figure 9: Percent of file written for write-only opens (cumulative)

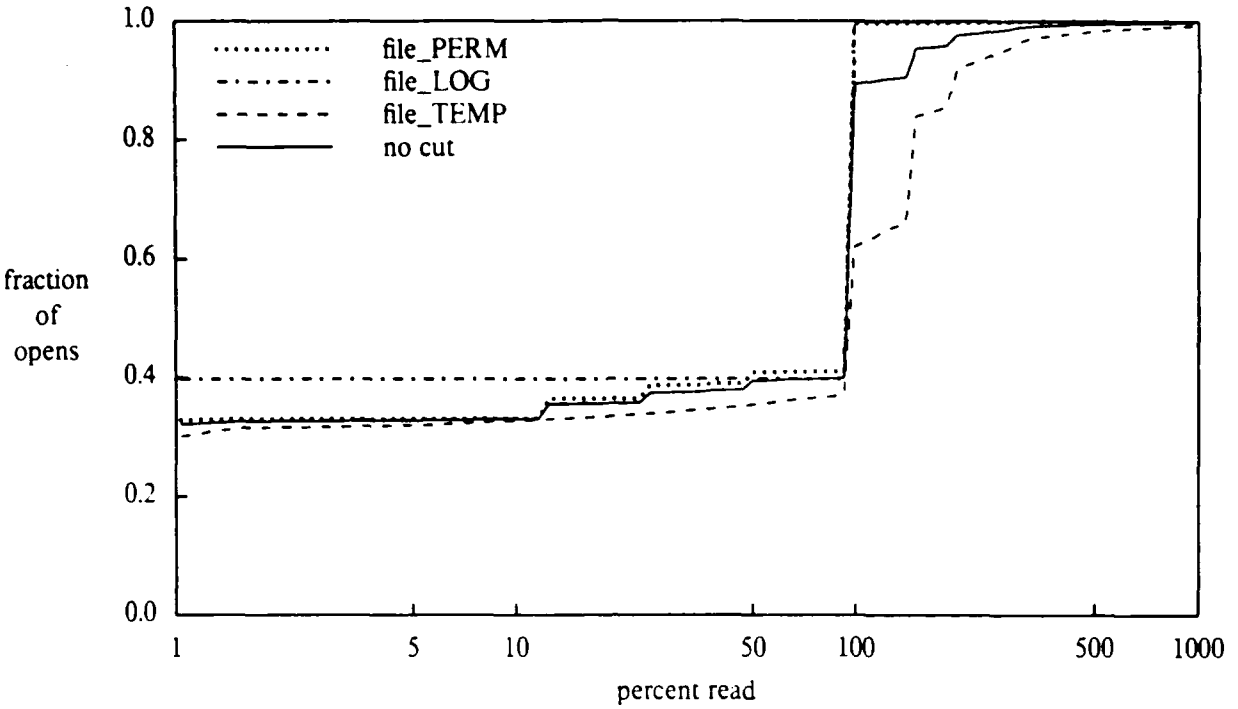


Figure 10: Percent of file read for read/write opens (cumulative)

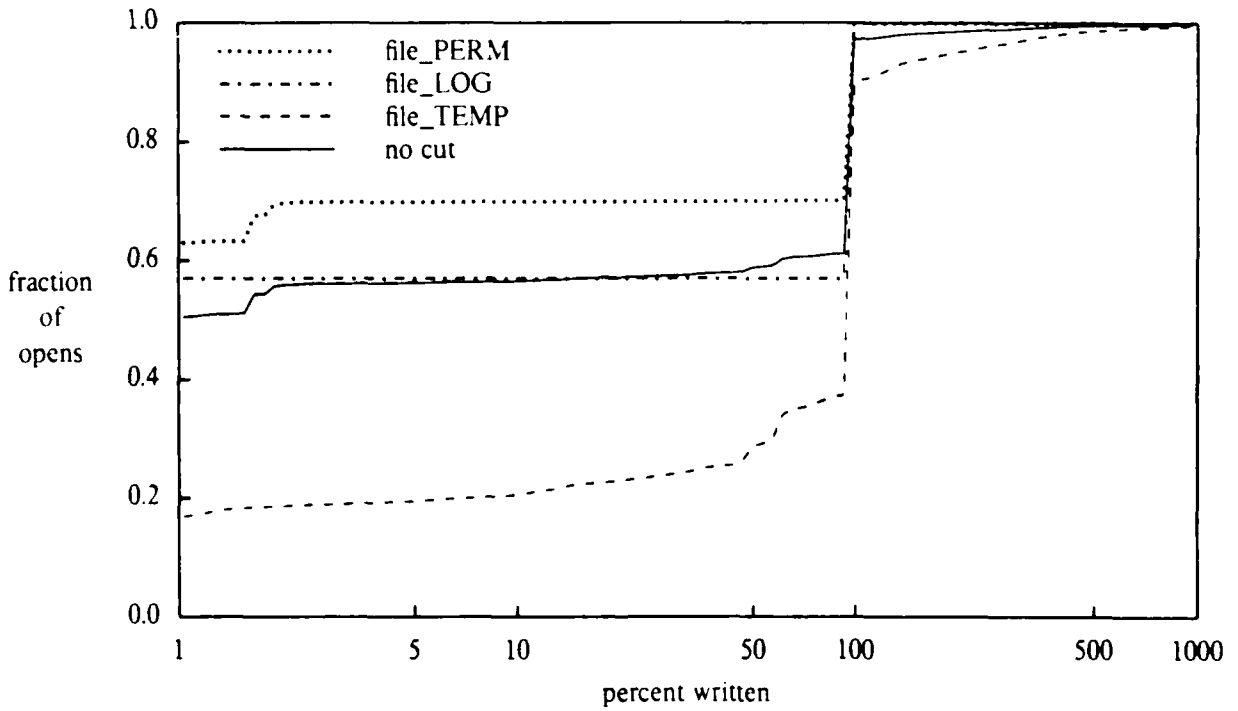


Figure 11: Percent of file written for read/write opens (cumulative)

distribution	min	max	mean	median	std dev	<100%	>100%
file_LOG	0	690	85.8	100	72	26%	3.3%
file_PERM	0	64100	83.2	100	235	36%	3.2%
file_TEMP	0	3600	85.8	100	53	18%	2.1%
no cut	0	64100	83.9	100	202	31%	2.9%

Table 11: Percentage read (read-only opens)

distribution	min	max	mean	median	std dev	<100%	>100%
file_LOG	0	100	2.8	<1	12	98.8%	0%
file_PERM	0	200	96.6	100	18	3.9%	0%
file_TEMP	0	9600	100.8	100	85	0.7%	0.2%
no cut	0	9600	85.7	100	53	15%	0.1%

Table 12: Percentage written (write-only opens)

distribution	min	max	mean	median	std dev	<100%	>100%
file_LOG	0	100	60.2	100	49	40%	0%
file_PERM	0	1900	62.5	100	66	41%	0.4%
file_TEMP	0	65000	138	100	1180	37%	37%
no cut	0	65000	82.9	100	615	40%	11%

Table 13: Percentage read (read/write opens)

distribution	min	max	mean	median	std dev	<100%	>100%
file_LOG	0	100	43.0	<1	49.5	57%	0%
file_PERM	0	20000	36.4	<1	275	70%	0.1%
file_TEMP	0	3600	93.5	100	150	37%	9.8%
no cut	0	20000	51.8	<1	249	61%	2.7%

Table 14: Percentage written (read/write opens)

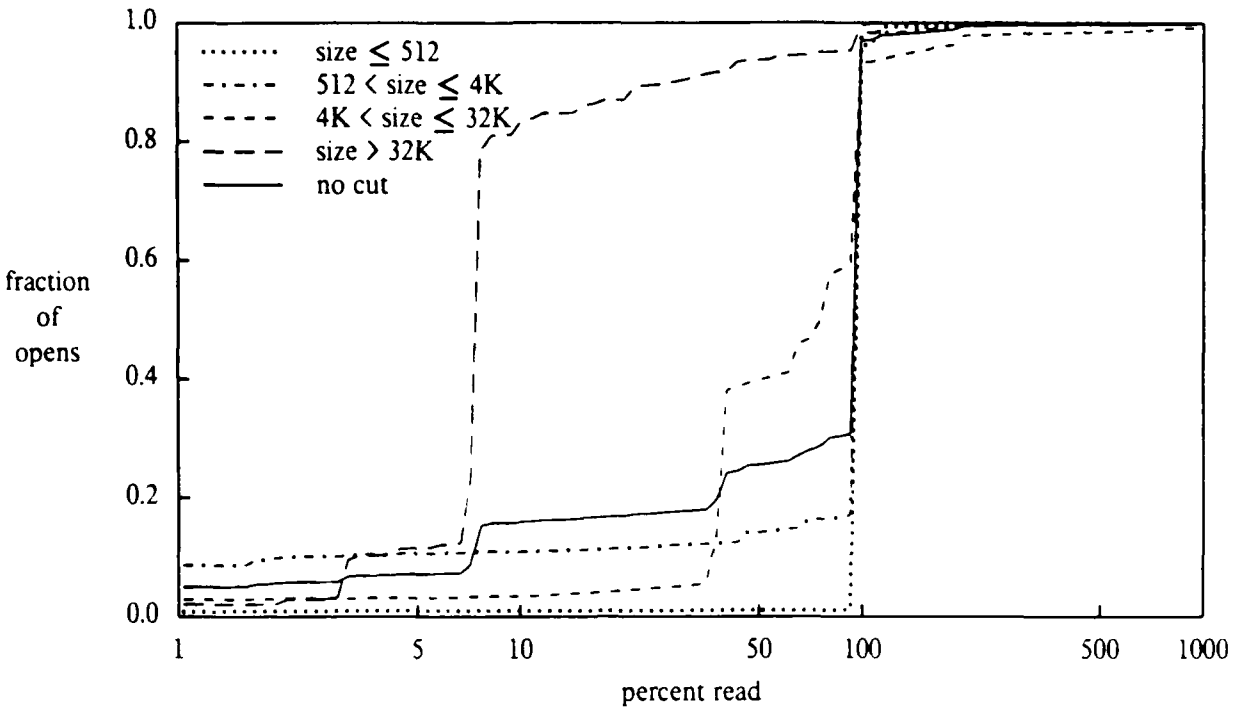


Figure 12: Percent of file read for read-only opens (cumulative, by size)

size	fraction of r-o opens	min	max	mean	median	std dev	<100%	>100%
≤ 512 bytes	22.7%	0	3600	100.1	100	19	1.2%	3.8%
512 < size ≤ 4K	46.7%	0	64100	88.4	100	158	17%	1.7%
4K < size ≤ 32K	18.5%	0	55500	97.0	80	382	59%	6.7%
size > 32K bytes	12.1%	0	12500	15.6	7.7	110	95%	0.4%
all	100%	0	64100	83.9	100	202	31%	2.9%

Table 15: Percentage read, by size (read-only opens)

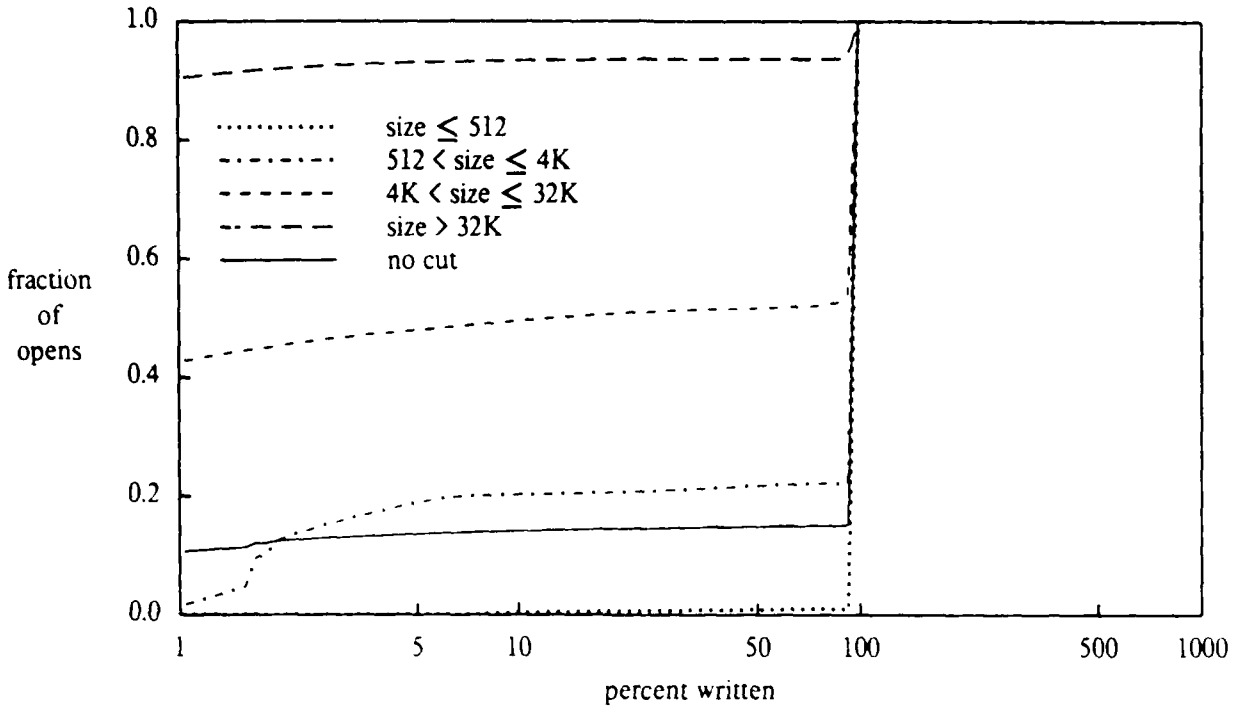


Figure 13: Percent of file written for write-only opens (cumulative, by size)

size	fraction of w-o opens	min	max	mean	median	std dev	<100%	>100%
≤ 512 bytes	71.5%	0	6800	99.5	100	35	1.0%	0.1%
512 < size ≤ 4K	13.0%	0	9600	79.7	100	84	22%	0.3%
4K < size ≤ 32K	7.3%	0	101	48.8	12	49	54%	0%
size > 32K bytes	8.2%	0	100	6.5	<1	24	94.2%	0%
all	100%	0	9600	85.7	100	53	15%	0.1%

Table 16: Percentage written, by size (write-only opens)

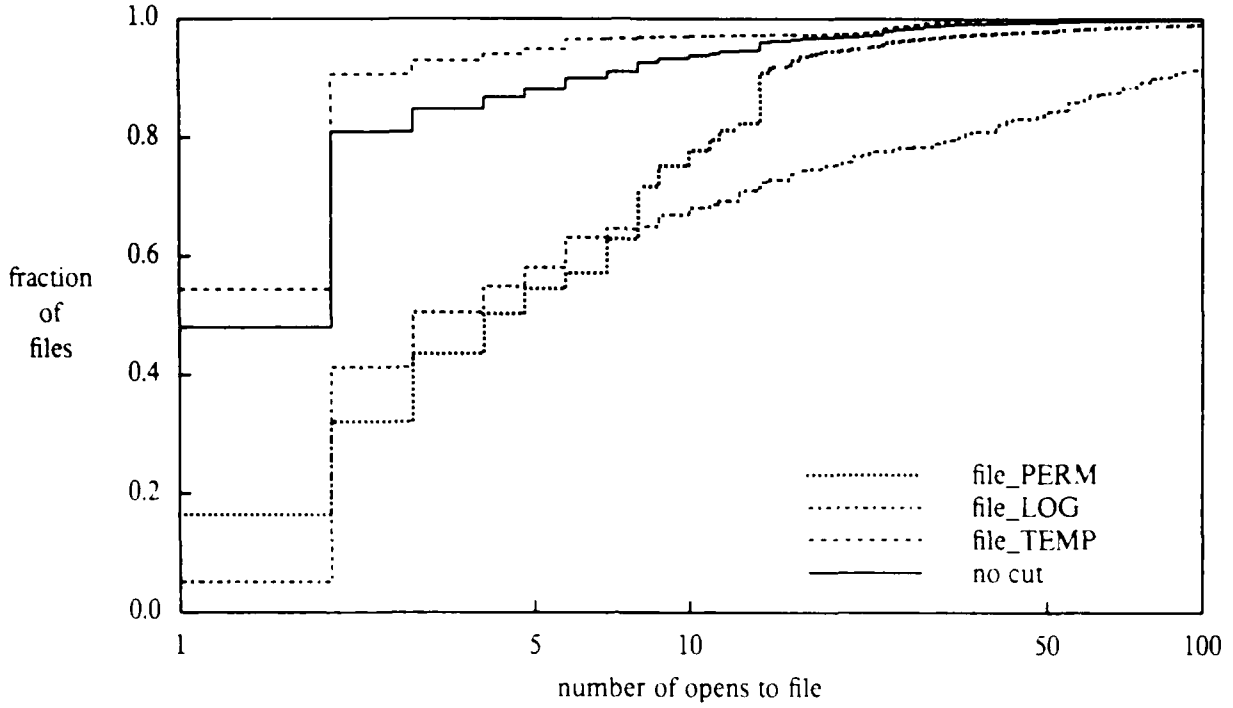


Figure 14: Number of opens per active file (cumulative)

distribution	mean	median	opened once	opened twice	opened more than twice	max
file_LOG	70.5	3	5%	36%	59%	5330
file_PERM	30.5	4	16%	16%	68%	26800
file_TEMP	2.6	1	55%	36%	9%	1920
no cut	7.5	2	48%	33%	19%	26800
SLAC, disk file_PERM	12	2	46%	23%	31%	2660

Table 17: Number of opens/file

6.1.3. Per File Results

The number of opens per file gives an indication of the potential benefits and penalties of migrating files to a user's machine (the degree of sharing is also a factor here). Most files in our environment were opened only once or twice (figure 14 and table 17). This may be attributed to the large number of lightly used temp files; log and perm files saw considerably more activity. The low number of opens for most files suggests that the initial placement of a file is an important consideration. We have also included in table 17 information on the distribution for on-disk permanent files in the SLAC trace (for a period of 310 hours). SLAC perm files saw, on average, considerably less activity than the perm files in our environment, despite the longer SLAC logging period.

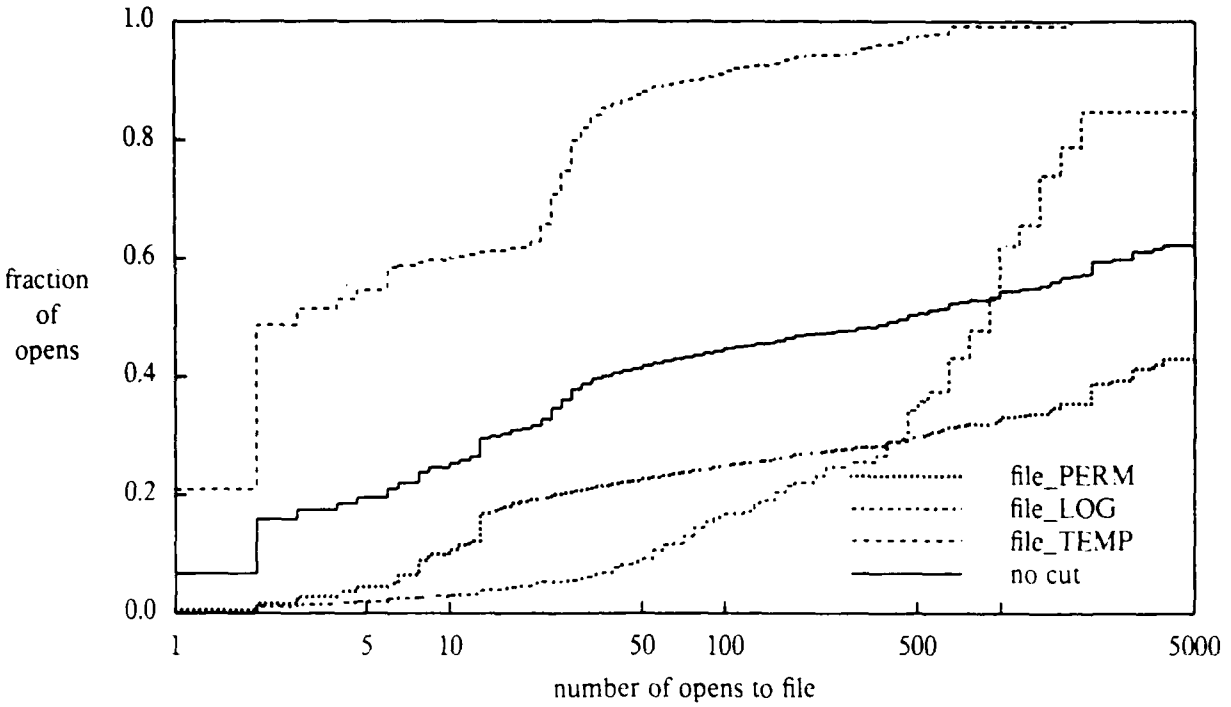


Figure 15: Fraction of opens per active file (cumulative)

distribution	mean	median	std dev
file_LOG	1480	950	1700
file_PERM	8120	>5000	8100
file_TEMP	57	3	210
no cut	5400	480	7600

Table 18: Open distribution (as a function of opens/file)

Most opens went to files opened many times. 75% went to files opened more than 10 times and half to files opened more than 480 times (figure 15 and table 18). For these files (and hence for a distributed file system as a whole), migration and/or replication may be useful. This will be especially true for perm files, since they receive so much open activity per file and tend to be opened read-only.

The most frequently opened files on Seneca were administrative and configuration files in /etc and in library directories, rwho daemon files and news databases. A table of the most frequently opened files may be found in appendix A.

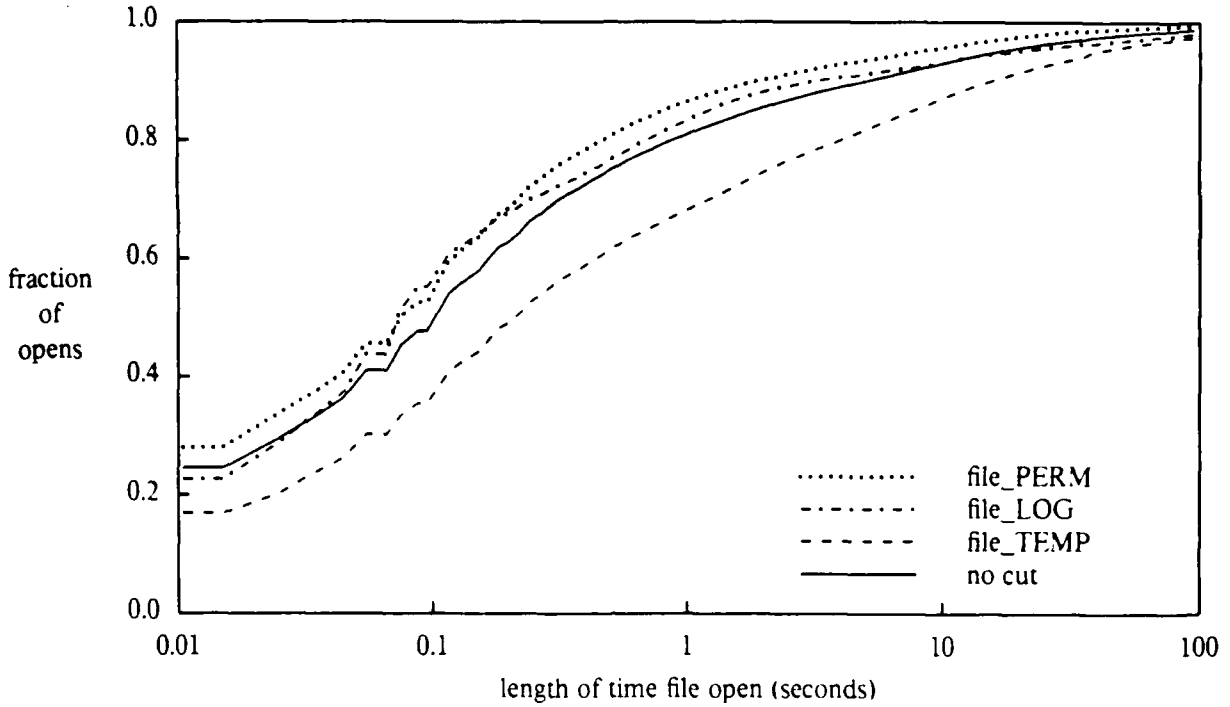


Figure 16: Times from file open to close (cumulative)

distribution	min	max	mean	median	std deviation
file_LOG	0	8.6e4	33.4	0.08	1140
file_PERM	0	7.6e4	6.5	0.08	251
file_TEMP	0	4.8e4	20.5	0.22	335
no cut	0	8.6e4	11.8	0.1	369

Table 19: Open time (seconds)

Files in our environment were usually only open for a few tenths of a second (figure 16 and table 19). Temp files were open for relatively long periods of time. This is to be expected, since they are often used to store intermediate results as they are being calculated. The distribution for perm files is consistent with the small files sizes and whole file transfers we saw earlier. Programs open these files, transfer data and then immediately close the files.

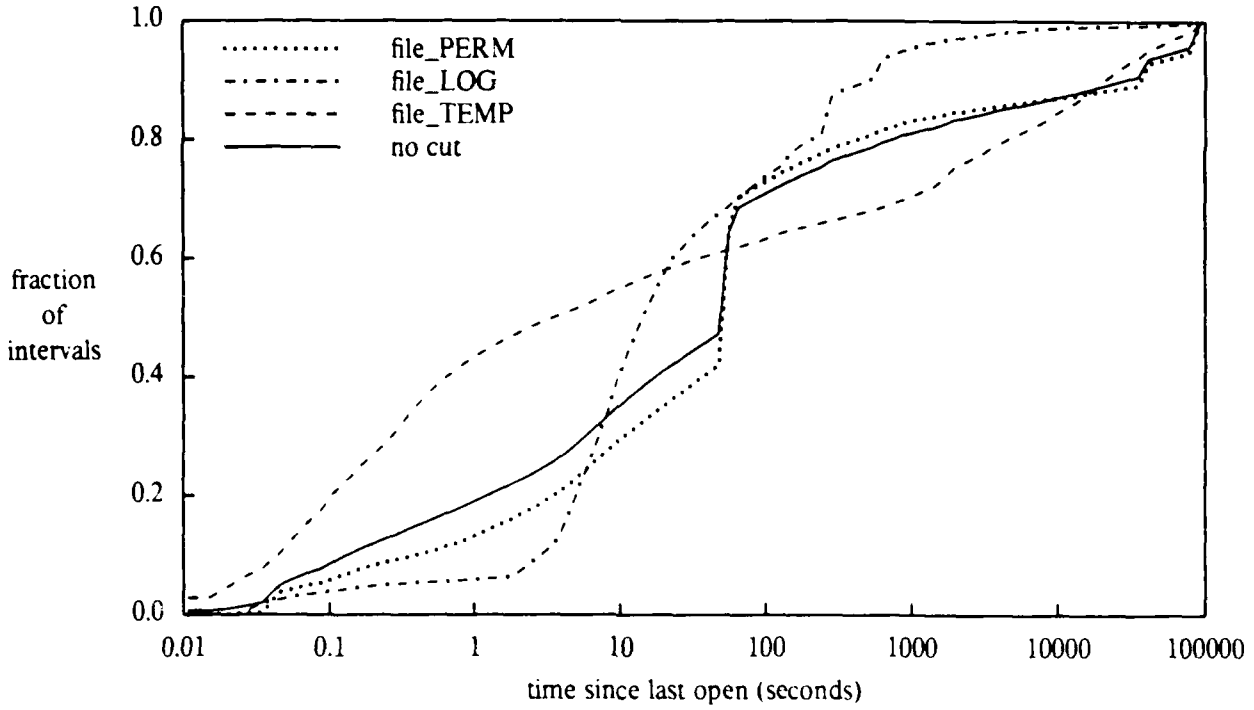


Figure 17: File interopen intervals (cumulative)

distribution	min	max	mean	median	std deviation
file_LOG	0	5.4e5	965	15	9.8e3
file_PERM	0	5.4e5	8215	60	2.4e4
file_TEMP	0	4.2e5	6655	3.6	1.8e4
no cut	0	5.4e5	7502	60	2.2e4
SLAC, disk file_PERM	0	9.7e4	8350	50	4.4e4

Table 20: File interopen intervals (seconds)

Knowledge of file interopen intervals (the time from one open of a file to the next) is useful in estimating both the appropriate time scale for migration and the possibilities for caching. Figure 17 and table 20 show that interopen intervals in our environment were short (opens to a file were strongly clustered). When a file was opened, the following open (if any) had a 50% probability of occurring within the next 60 seconds. Interopen intervals for temp files were particularly short. If a temp file was opened multiple times (many were not), the next open often occurred within a few seconds of the last one. This is to be expected for files that are used to hold results between job steps. Log files also had shorter interopen intervals than files as a whole. Most log file opens were made by net processes and these processes show intense bursts of activity (figure 2), so this is not surprising. The jump at 60 seconds in the distribution for perm files is due to rwho daemon activity.

The lifetime of a file in our environment depended strongly on the class of the file. Most temp files lived less than a minute. The overwhelming majority of perm files had lifetimes that extended beyond the logging period. Log files fell in between (mostly due to short lived UUCP work logs). File lifetime distributions are shown in figure 18. Here files that existed before logging was started or that continued to exist after logging was terminated were given lifetimes exceeding the logging period (lie to the right of the histogram). Because so many log and perm files fell into this category, we have not included the moments of these distributions.

Even though most perm files have long happy lifetimes, the data in these files is not so fortunate. This is shown in figure 19, where we have histogrammed the time from when a file is created or written to the time when a file is overwritten or deleted (this is the file lifetime used by Ousterhout et al.). Files that were only partially written are not included in this histogram. Again, data whose lifetime extended beyond the limits of our log were given lifetimes exceeding the logging period. The large jump at 60 seconds is due to rwho daemon activity. Since we include all files here and Ousterhout et al. included just new data, our results are not directly comparable.

The first two columns of table 21 show the mean number of readers per file, as indicated by the account (ruid) of the reader, and the percentage of files with more than one reader, cut by the file class and owner. The next four columns show this information for writers and for the overall number of file users. The last two columns show the mean and maximum number of *inversions* per file. The number of inversions is the number of times that the most recent user of the file changes (this is basically the inversion clustering metric used by Porcar [Porcar 82]). For a file used by only one user, the number of inversions will be zero.

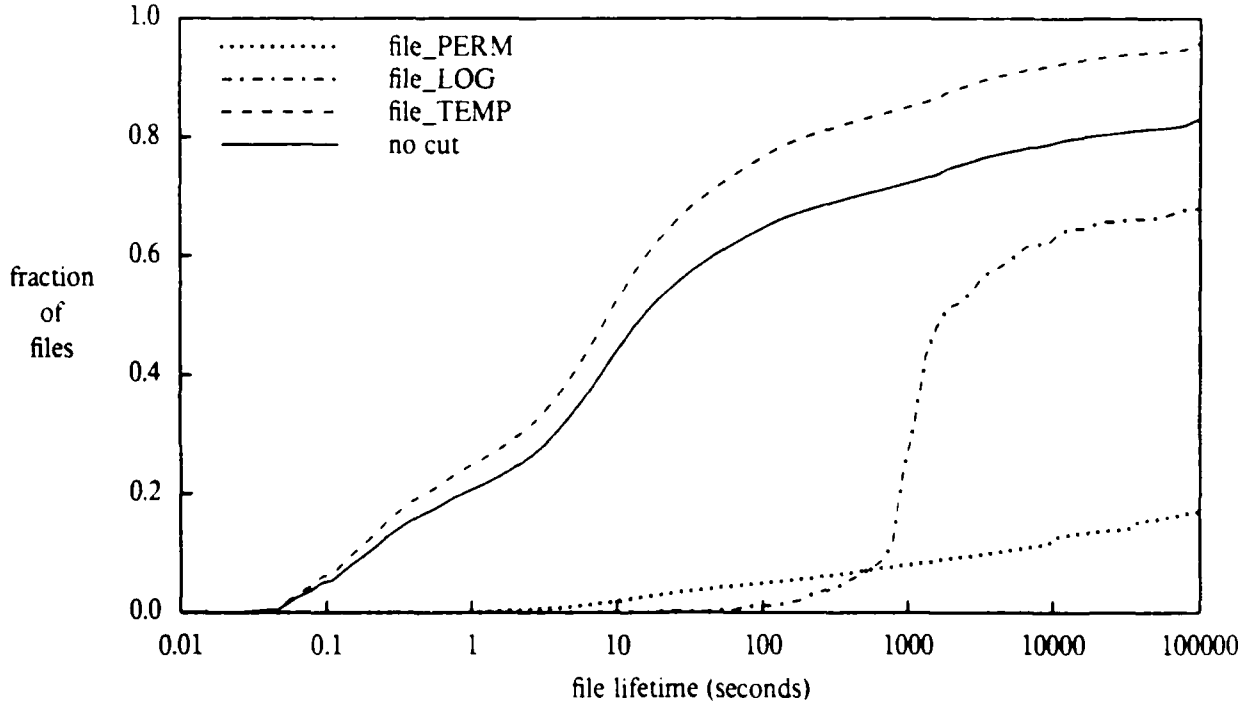


Figure 18: File lifetimes (cumulative, files living beyond log period binned at right)

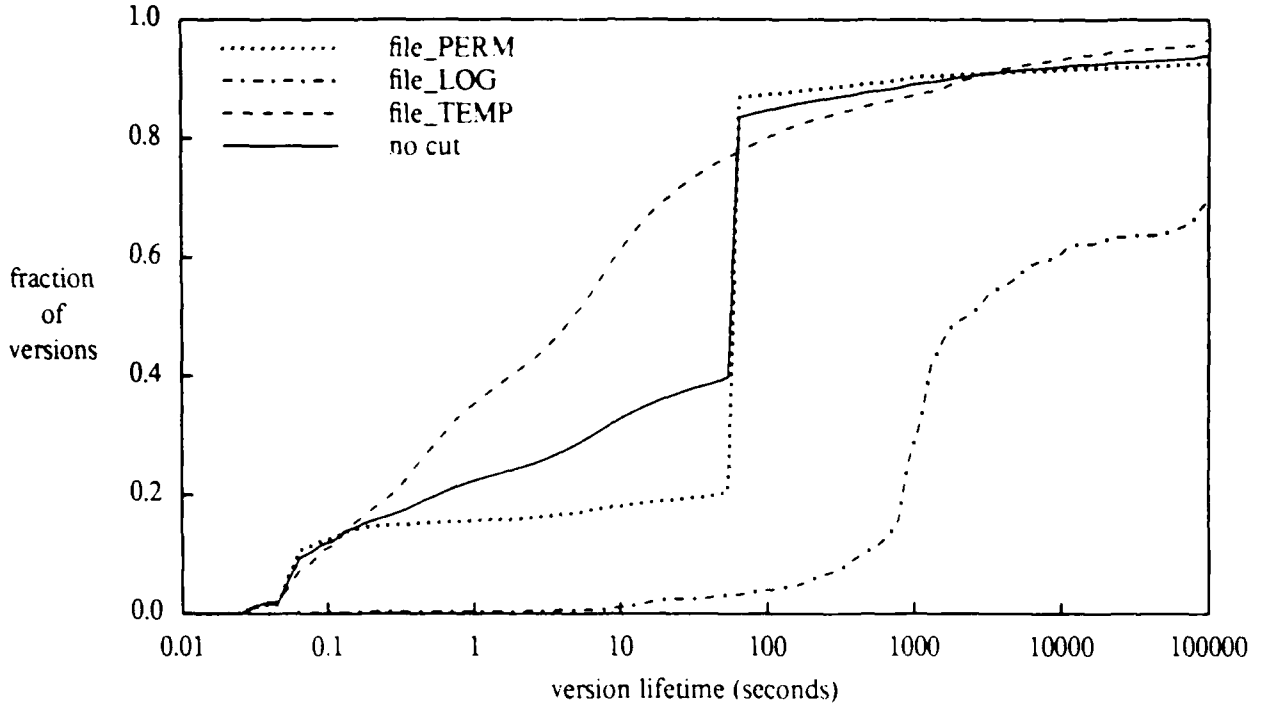


Figure 19: Version lifetimes (cumulative, versions living beyond log period binned at right)

We can see from table 21 that 11.8% of the files seen during the logging period were accessed by multiple users (users with separate accounts). Multiple readers were much more common than multiple writers. Most shared files belonged to net. These were predominately news articles (perm files). Logs were also heavily shared. They frequently had multiple writers and separate readers. Although system files were not as heavily shared as net files, in terms of the number of shared files, the high mean number of inversions (2.92) indicates that the system files that were shared were not shy about it. Few user files were shared. The low mean number of inversions (0.111) indicates that this sharing was incidental to the normal use of user files.

cut	readers		writers		users (r | w)		inversions	
	mean	>1	mean	>1	mean	>1	mean	max
file_LOG	0.98	3.8%	1.86	26.7%	2.67	76.9%	5.25	293
file_PERM	1.575	17.9%	0.444	3.4%	1.711	20.9%	5.52	12529
file_TEMP	0.639	4.4%	1.02	2.1%	1.212	9.6%	0.293	92
owner_NET	0.905	11.9%	0.933	4.1%	1.501	20.7%	0.874	1288
owner_SYSTEM	0.483	3.0%	0.962	0.12%	1.196	3.8%	2.92	12529
owner_USER	0.85	1.3%	0.876	0.84%	1.053	2.7%	0.111	169
no cut	0.792	6.6%	0.930	2.4%	1.30	11.8%	1.16	12529

Table 21: file sharing, by file class and owner

The overall distributions are shown in more detail in table 22. Note that very few files had more than 2 writers and that even the distribution of the number of users per file drops off quite sharply.

number	readers		writers		users (r | w)		inversions	
	count	cum	count	cum	count	cum	count	cum
0	44711	44.2%	11252	11.1%	-	-	89272	88.2%
1	49845	93.4%	87510	97.6%	89272	88.2%	5838	94.0%
2	3022	96.4%	2009	99.59%	7555	95.7%	2182	96.2%
3	1244	97.7%	209	99.80%	1818	97.5%	666	96.8%
4	685	98.3%	66	99.86%	758	98.2%	799	97.6%
5	421	98.8%	23	99.89%	448	98.7%	389	98.0%
6	356	99.11%	18	99.90%	369	99.05%	339	98.3%
7	245	99.35%	20	99.92%	258	99.30%	318	98.6%
8	167	99.52%	8	99.93%	175	99.47%	291	98.9%
9	80	99.60%	12	99.94%	88	99.56%	213	99.13%
10	105	99.70%	8	99.95%	111	99.67%	156	99.29%
>10	304	100%	50	100%	333	100%	722	100%
total	101185	-	101185	-	101185	-	101185	-

Table 22: readers, writers, users and inversions; no cuts

6.2. Execute Patterns

The basic calls to run an executable file under 4.2BSD UNIX are execv and execve. These calls are grouped together under the heading "execute" in table 3. Users were responsible for half of the execute requests in our log (table 23), even though, as we saw in section 6.1, they made only a quarter of the opens to regular files. Most executes were done on system files. Users owned almost half of the executables seen but there were few executes of these files.

cut	executes	% executes	executables	% executables	executes/executable
ruid_NET	26761	21.4%	41	7.1%	653
ruid_SYSTEM	38093	30.5%	137	23.6%	278
ruid_USER	60210	48.1%	528	90.9%	114
owner_NET	12190	9.7%	34	5.9%	359
owner_SYSTEM	108646	86.9%	291	50.1%	373
owner_USER	4228	3.4%	256	44.1%	17
no cut	125064	100%	581	100%	215

Table 23: Basic active executable statistics

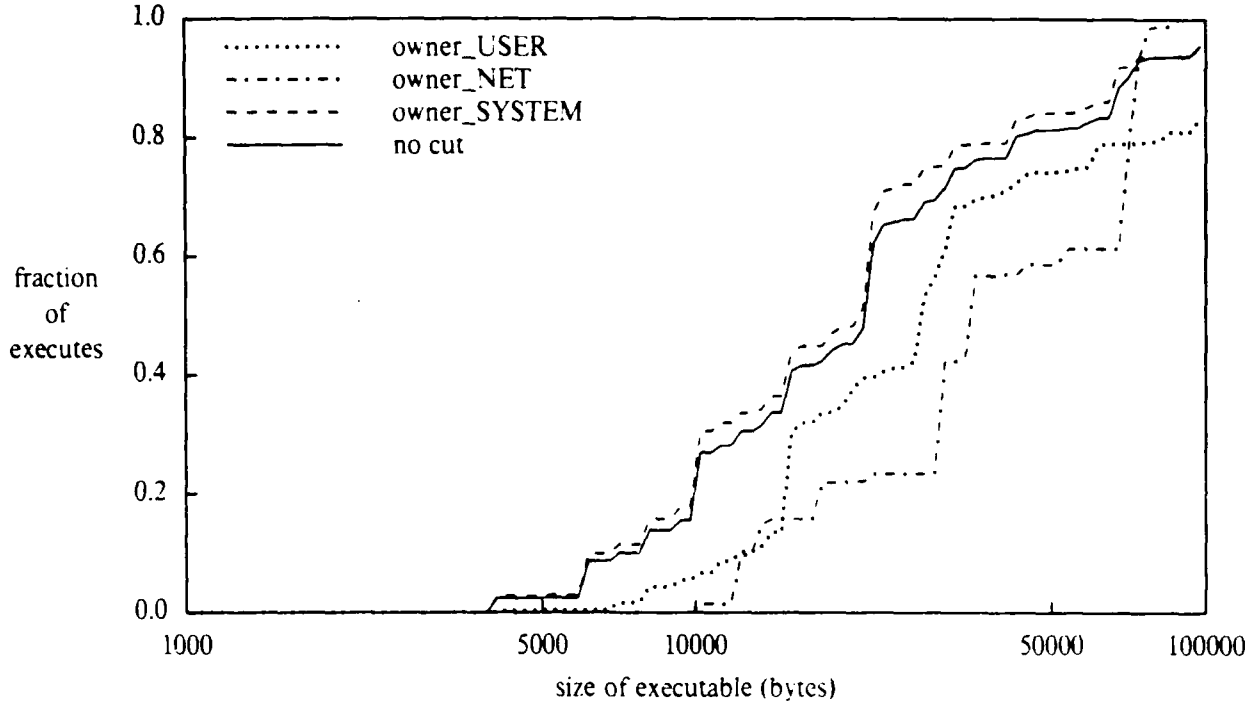


Figure 20: Dynamic executable file size distributions (cumulative)

distribution	min	max	mean	median	std deviation
owner_NET	9216	8.8e4	44400	35500	23900
owner_SYSTEM	4096	1.1e6	34500	21400	84900
owner_USER	4228	3.2e6	55900	28200	135000
no cut	4096	3.2e6	36200	22400	83400

Table 24: Executable file sizes (bytes)

Most executable files were between 5,000 and 100,000 bytes long (figure 20 and table 24). The relatively large size of executables is a reflection of the lack of run-time library sharing. All executables contain whatever code they need to run.

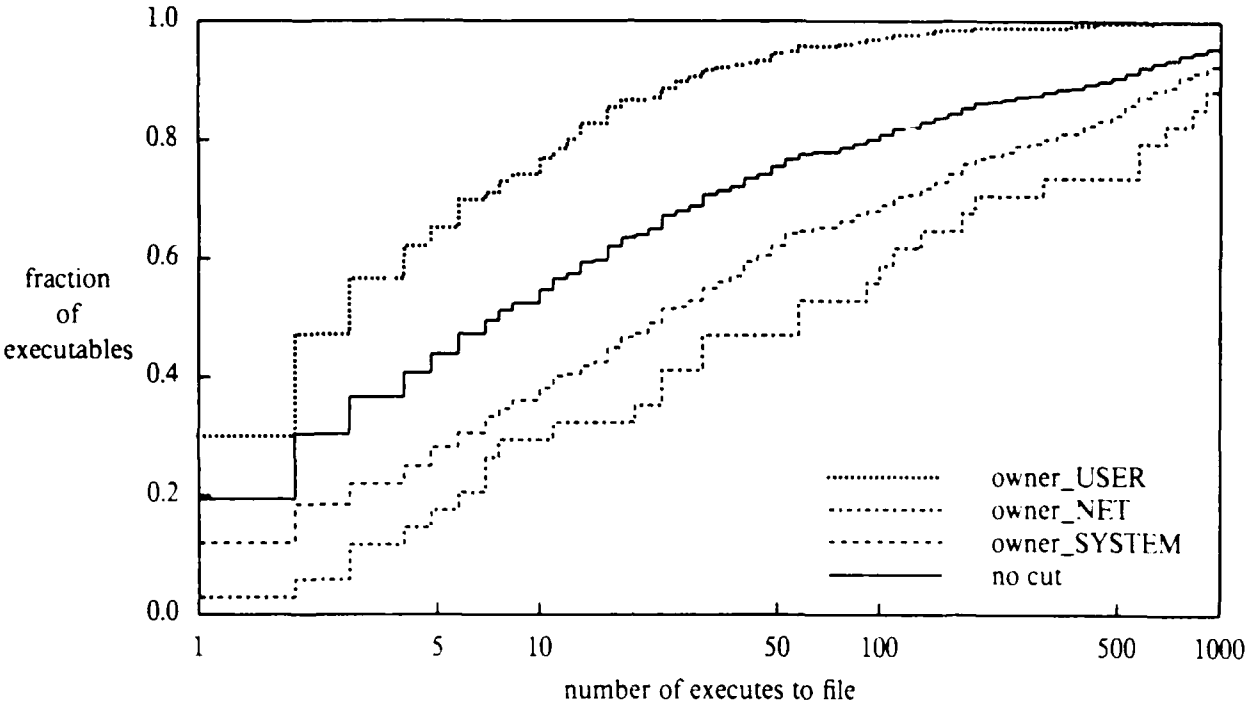


Figure 21: Number of executes per active executable (cumulative)

distribution	min	max	mean	median	std deviation
owner_NET	1	2152	359	60	570
owner_SYSTEM	1	17519	373	24	1340
owner_USER	1	675	17	3	59
no cut	1	17519	215	8	970

Table 25: Number of executes/active executable

An executable file saw considerably more activity than other regular files (figure 21 and table 25). Almost half were executed 10 times or more. This is not surprising, considering the small number of active executables.

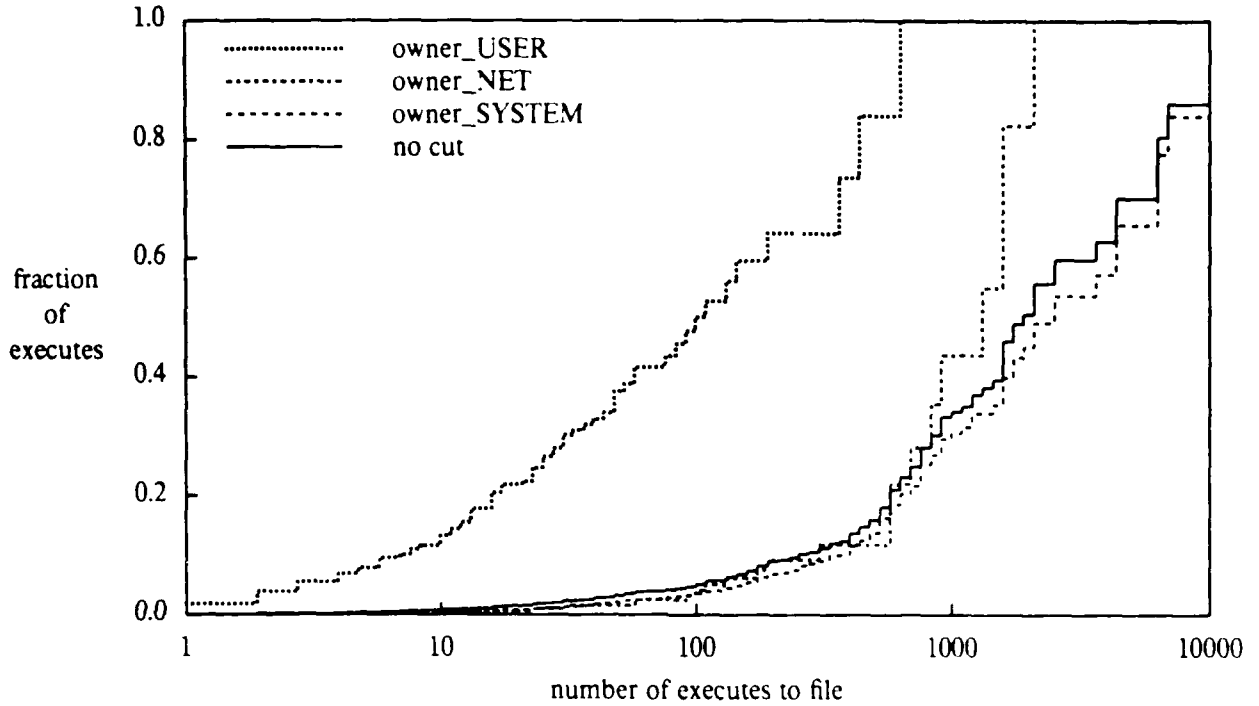


Figure 22: Fraction of executes per active executable (cumulative)

distribution	mean	median	std dev
owner_NET	1270	1380	634
owner_SYSTEM	5150	2600	5850
owner_USER	230	105	244
no cut	4600	2000	5640

Table 26: execute distribution (as a function of executes/executable)

Most executes went to files executed a large number of times. Half went to files executed more than 2000 times (figure 22 and table 26) and 95% went to files executed at least 100 times.

The most frequently executed files on Seneca were shells and system utilities to delete files, evaluate conditionals, list directories and distribute files to other machines (table A-2 in appendix A). Over half of the executes went to only 13 files. These files, taken together, occupied 0.46MB of disk space (0.08% of the total). This suggests that even a very modest amount of caching or other special treatment for frequently requested programs will produce significant improvements. Evidence for this was also seen in a study of

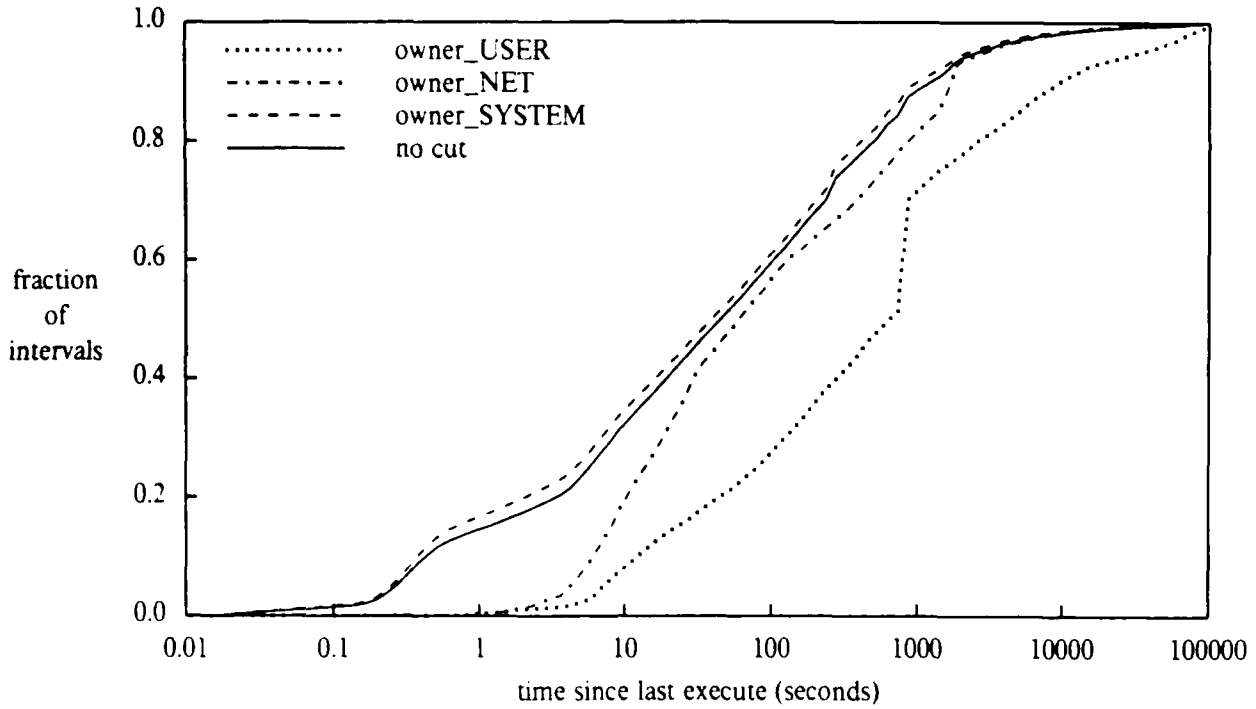


Figure 23: File interexecute intervals (cumulative)

distribution	min	max	mean	median	std deviation
owner_NET	0.05	2.2e5	1160	65	6550
owner_SYSTEM	0	5.6e5	983	40	7680
owner_USER	0.52	4.1e5	6290	730	23600
no cut	0	5.6e5	1170	47	8610

Table 27: Interexecute intervals (seconds)

2MB diskless Sun workstations running a version of UNIX similar to the one on Seneca at the University of Washington [Lazowska 84]. For the Suns studied, 80% of the bytes transferred were due to file accesses and only 20% were for paging. If we take our average executable size times the execute rate (most 4.2BSD executables are loaded using demand paging), we get a very crude paging estimate of 7500 bytes/second, or about 170% of the transfers due to opens (table 8). The difference between our crude estimate and the behavior seen at the University of Washington is probably due to both the caching of pages of frequently executed files and to code and debugging information in executables that is not used.

The distribution of time between executes for executables is given in figure 23 and table 27. These distributions lend support to our caching arguments (at least for selected executables owned by net and system).

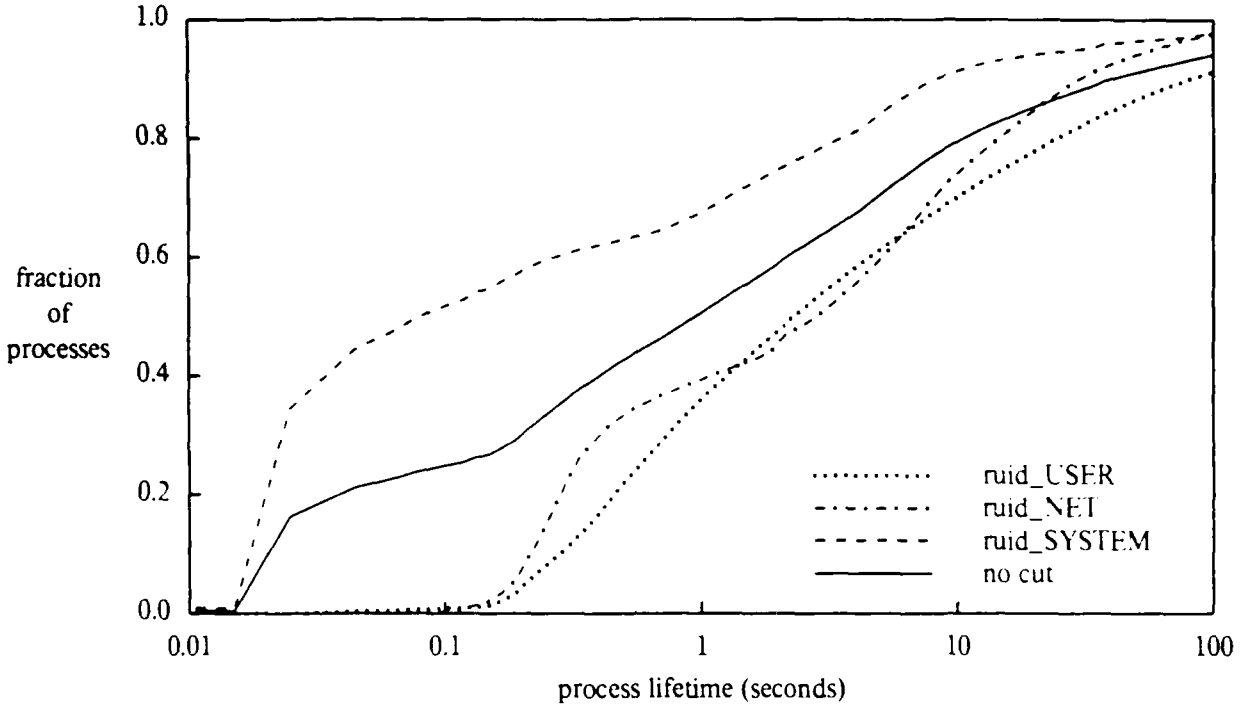


Figure 24: Process lifetimes (cumulative)

distribution	min	max	mean	median	std deviation
ruid_NET	0.02	7250	15.7	3.0	88
ruid_SYSTEM	0.01	215000	52.2	0.09	1380
ruid_USER	0.02	76100	118	2.4	1190
no cut	0.01	215000	165	0.95	2560

Table 28: Process lifetimes (seconds)

Executing a program on UNIX is usually done using the sequence fork (to create a copy of the running process); execv or execve (to replace that copy with the new program); exit (when done). Since over 2/3 of the forks on Seneca were followed by an execute we can, by looking at process lifetimes (time from fork to exit) estimate how long executables were in use. Process lifetime distributions, cut by the ruid of the requester, are given in figure 24 and table 28[6]. Over half of all processes recorded in the log lived less than a second. System processes were particularly short-lived. With the exception of the large number of system processes that lived less than a tenth of a second (due mostly to local network servers) our results agree with process lifetime results given by Zhou et al. [Zhou 85].

[6]Some processes, such as login shells, start life in one ruid class and exit in another. These are included only in the overall distribution.

Executable files were more heavily shared than opened files (table 29). This should come as no surprise, since there were relatively few executables and there were usually located in public directories. User executables were relatively lightly shared.

cut	executors					inversions			
	mean	median	>1	>5	max	mean	>0	>5	max
owner_NET	10.3	4	70.6%	41.2%	45	92.9	70.6%	55.9%	957
owner_SYSTEM	12.0	2	61.5%	34.4%	111	126	61.5%	38.8%	6539
owner_USER	1.38	1	9.8%	1.6%	28	1.74	9.8%	4.7%	205
no cut	7.2	1	39.2%	20.3%	111	69.5	39.2%	24.6%	6539

Table 29: Executable sharing

6.3. User File Patterns

In this section we take a closer look at user files. Many distributed file systems (including Roe [Ellis 83] and the ITC DFS [Satyanarayanan 85]) deal primarily or wholly with user files. In addition, we expect that user file access patterns will be less dependent on the operating system used. These factors make user file reference patterns particularly interesting.

The results presented in this section are actually for user references to user files (owner_USER + ruid_USER cut, referred to as the "U" cut below). These references represented over 90% of the references to user files. The remaining references were mostly infrequent periodic references made by system processes and had little effect on the distributions we see (with the exception of some of the sharing results). The organization of this section follows closely that of section 6.1.

6.3.1. Basic Statistics for User Files

The majority (62%) of user references to user files were to perm files (table 30), even though less than a third of the referenced user files were perm files. There were few references to log files. Most of these files were logs of mail sent or read and so the low level of activity is not surprising. With the exception of a somewhat higher proportion of perm files, these figures agree with what we saw for the overall distributions (table 5).

cut	opens	% opens	files	% files	opens/file
U + file_LOG	837	0.8%	101	0.3%	8.3
U + file_PERM	65051	62.4%	8662	29.0%	7.5
U + file_TEMP	38420	36.8%	21127	70.7%	1.8
U	104308	100%	29890	100%	3.5

Table 30: User opens to user files

cut	read-only		write-only		read/write		total
	opens	fraction	opens	fraction	opens	fraction	opens
U+file_LOG	117	14.0%	623	74.4%	97	11.6%	837
U+file_PERM	50193	77.5%	13296	20.5%	1310	2.0%	64799
U+file_TEMP	7349	19.1%	19891	51.8%	11140	29.0%	38380
U	57659	55.4%	33810	32.5%	12547	12.1%	104016

Table 31: Modes of open for user open-close sessions to user files

category	opens	% opens	files	% files	opens/file
library	2036	3.1%	91	1.1%	22.4
manual pages	776	1.2%	181	2.1%	4.3
program source	10538	16.2%	1486	17.1%	7.1
includes	3093	4.8%	306	3.5%	10.1
objects	5617	8.6%	467	5.4%	12.0
personal configuration	20278	31.2%	1638	18.9%	12.4
mail spool	2049	3.1%	453	5.2%	4.5
other	20644	31.8%	4040	46.6%	5.1

Table 32: Function of opened user perm files

55% of the opens were read-only with most of the read-only opens going to perm files (table 31). Users showed a strong tendency to open perm files read-only and other files write-only or read/write.

30% of the activity to perm files was to program development files ("program source," "includes," and "objects" in table 32). A similar number of references were to personal configuration files (often referred to as "dot files"). Most of the rest of the references were unidentifiable.

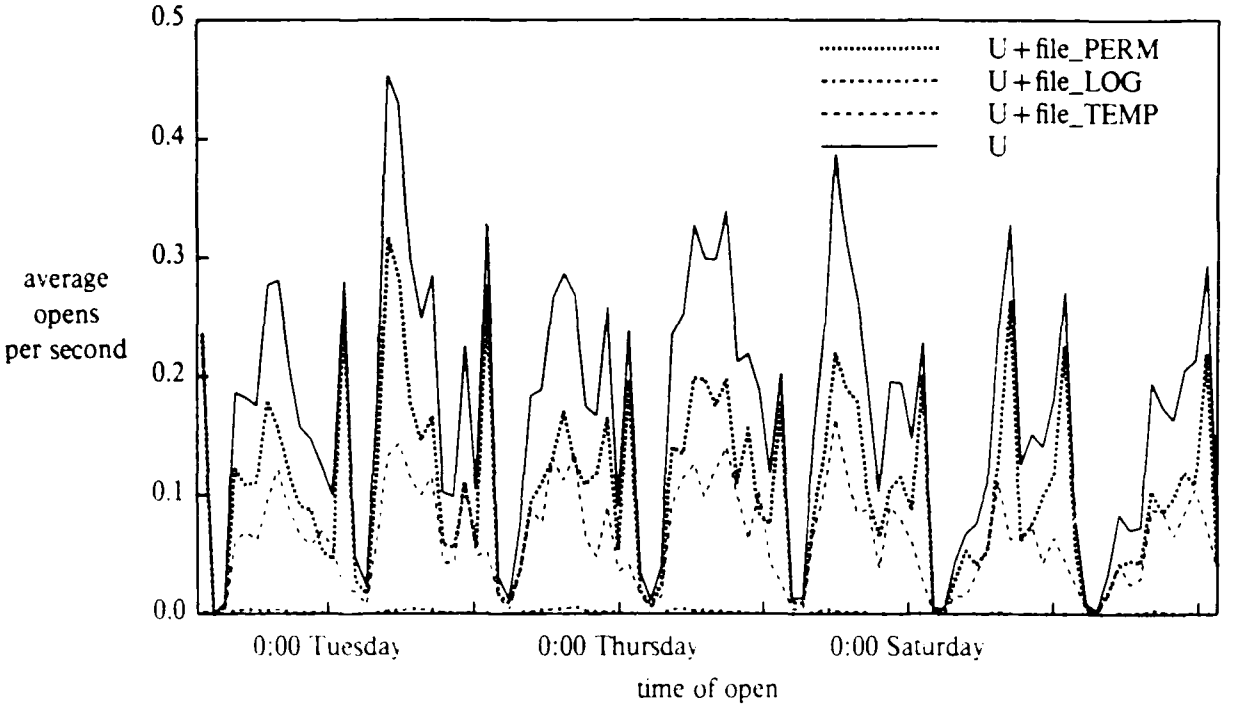


Figure 25: Average number of file opens per second (2 hour resolution, U cut)

cut	reads		writes		overall (r + w)	
	bytes/sec	fraction	bytes/sec	fraction	bytes/sec	fraction
U + file_LOG	9.6	1.0%	4.6	1.1%	14.2	1.0%
U + file_PERM	401	41.0%	121	28.6%	522	37.3%
U + file_TEMP	568	58.0%	297	70.4%	865	61.7%
U	978	100%	423	100%	1401	100%
no cut (table 8)	4190	-	800	-	4990	-

Table 33: Bytes read/written by users to user files

6.3.2. Per Open Results for User Files

User open activity to user files (figure 25) showed a busy period during the work day, with activity tapering off in the late evening. This is typical of a university environment. There was some early morning activity due to user background jobs. The overall level of activity was much less than what we saw for the system as a whole (user opens to user files accounted for 14% of the open activity) and was generally less bursty.

User reads and writes to user files accounted for 28% of the bytes transferred during the logging period. Most of the transfers (61.7%) were to and from temp files (table 33). Few bytes were transferred to or from log files.

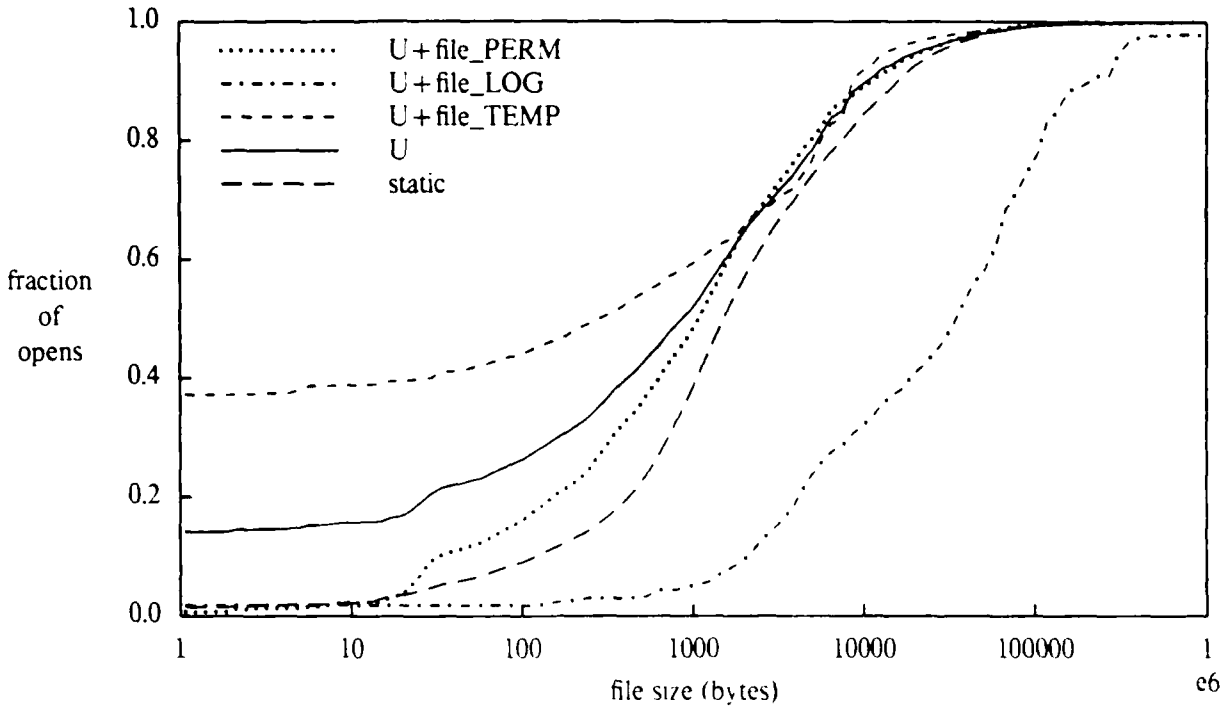


Figure 26: Dynamic file size distributions (cumulative, measured at close, U cut)

distribution	min	max	mean	median	std deviation
U+file_LOG, dynamic	0	1.28e6	90400	39000	1.7e5
U+file_PERM, dynamic	0	2.49e6	6205	1230	4.0e4
U+file_TEMP, dynamic	0	1.30e6	5006	310	2.4e4
U, dynamic	0	2.49e6	6440	930	3.9e4
all, dynamic (table 9)	0	2.49e6	18800	710	6.2e4
all, static	0	7.95e6	8020	1600	5.6e4

Table 34: User file size distributions

Cumulative file size distributions for users files, weighted by the number of user opens and cut by the file class, are given in figure 26 and table 34. Referenced user files were, on average, smaller than other referenced files. This was due, in part, to the large number of zero length temp files and to the absence of the large, frequently accessed administration files seen in the overall data.

Users accessed most of their files completely (figures 27-30 and tables 35-38). 90% of opens with read access (read-only or read/write) resulted in the file being completely read (compared to 68% for the system as a whole). 83% of files opened with write access were completely written (compared to 78% for the system as a whole). Nearly all files opened read-only were completely read.

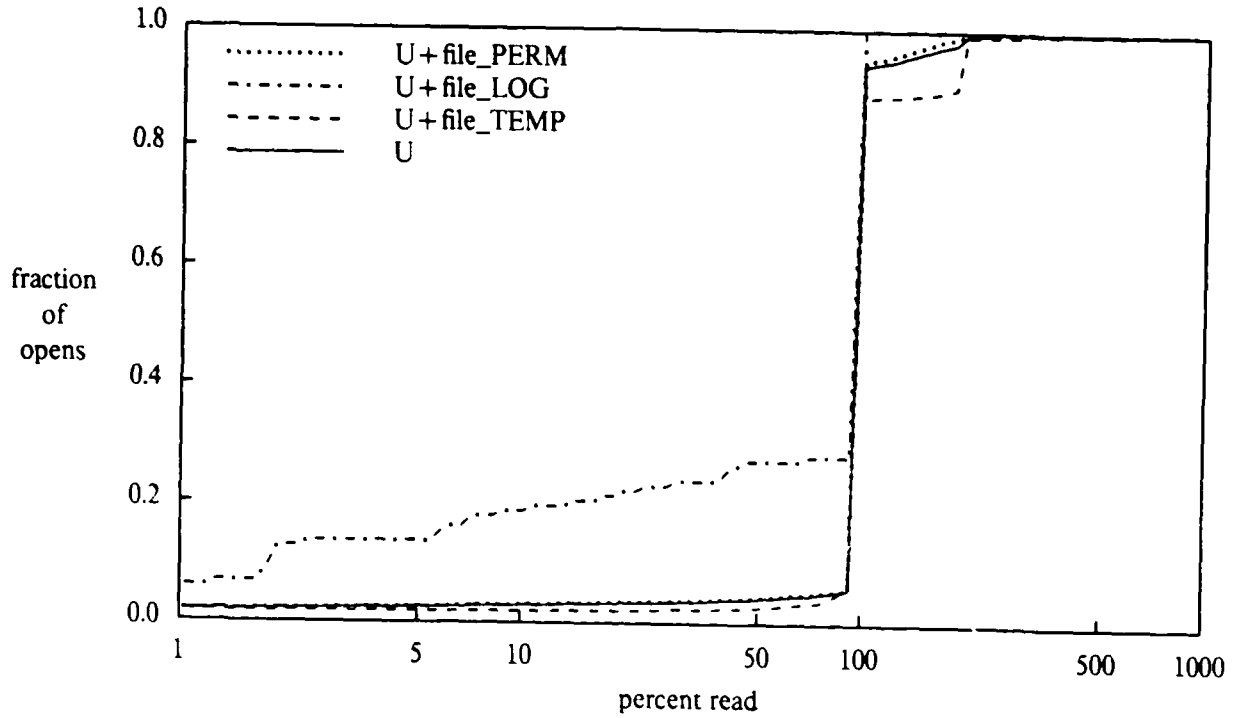


Figure 27: Percent of file read for read-only opens (cumulative, U cut)

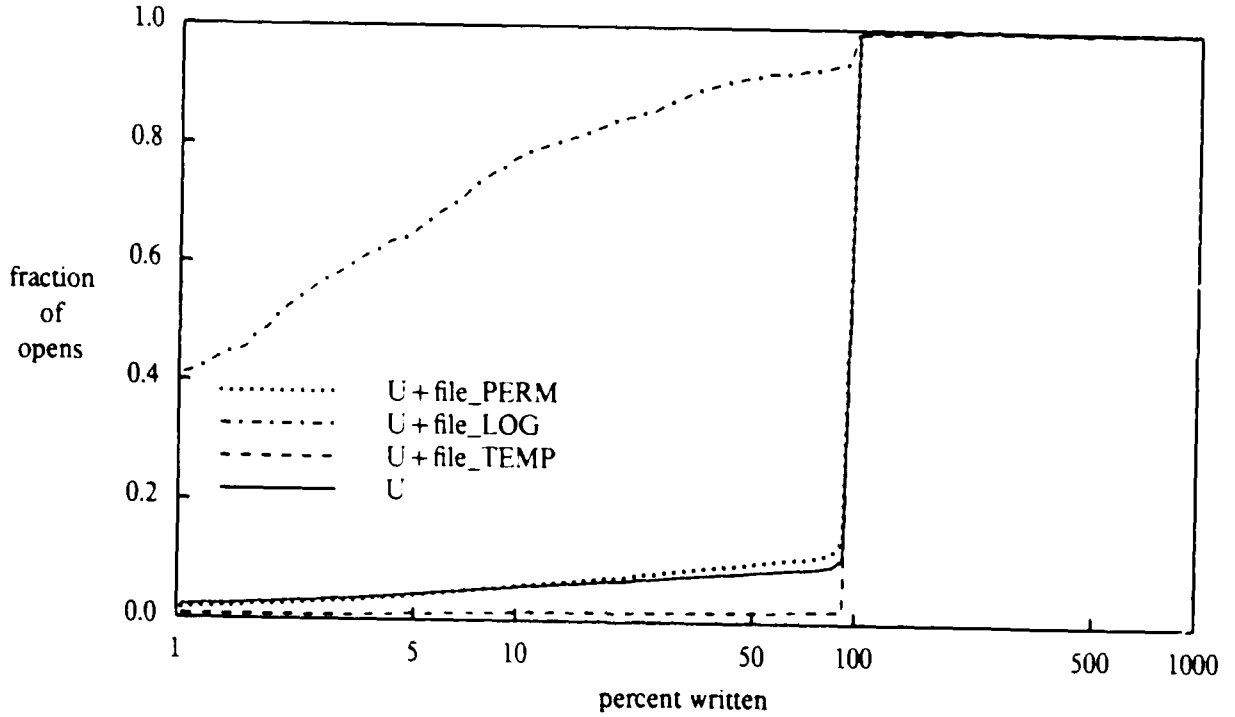


Figure 28: Percent of file written for write-only opens (cumulative, U cut)

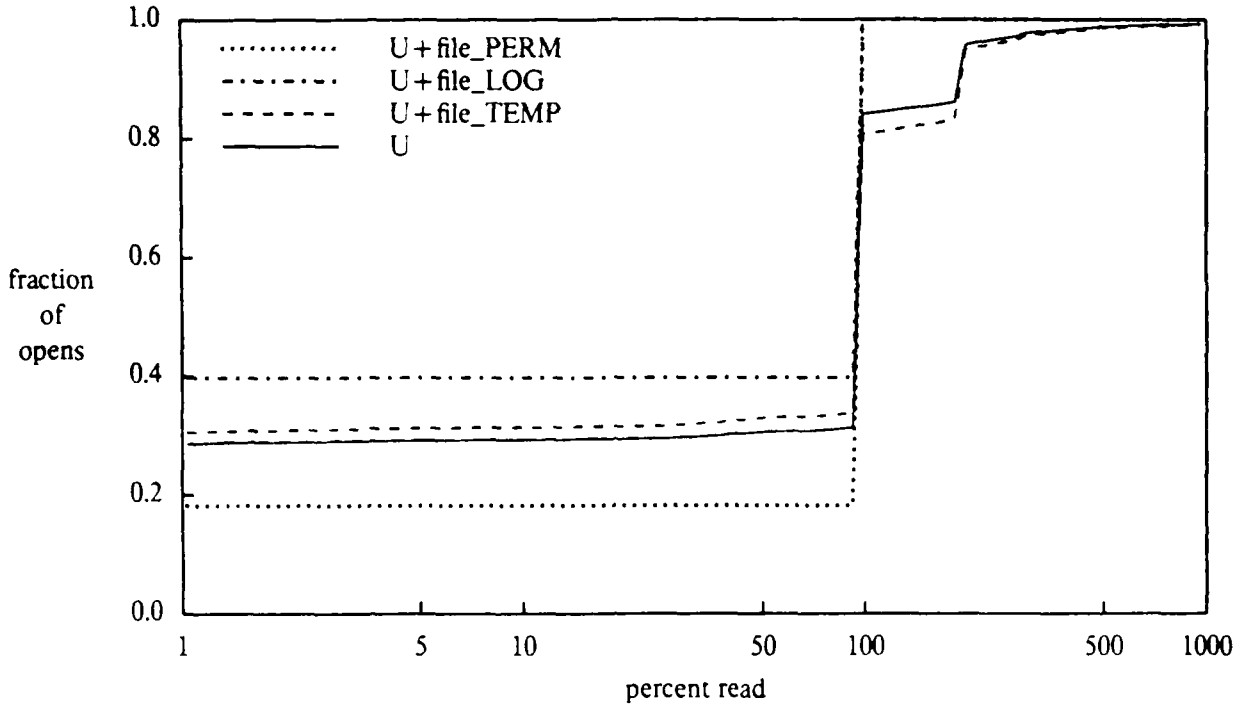


Figure 29: Percent of file read for read/write opens (cumulative, U cut)

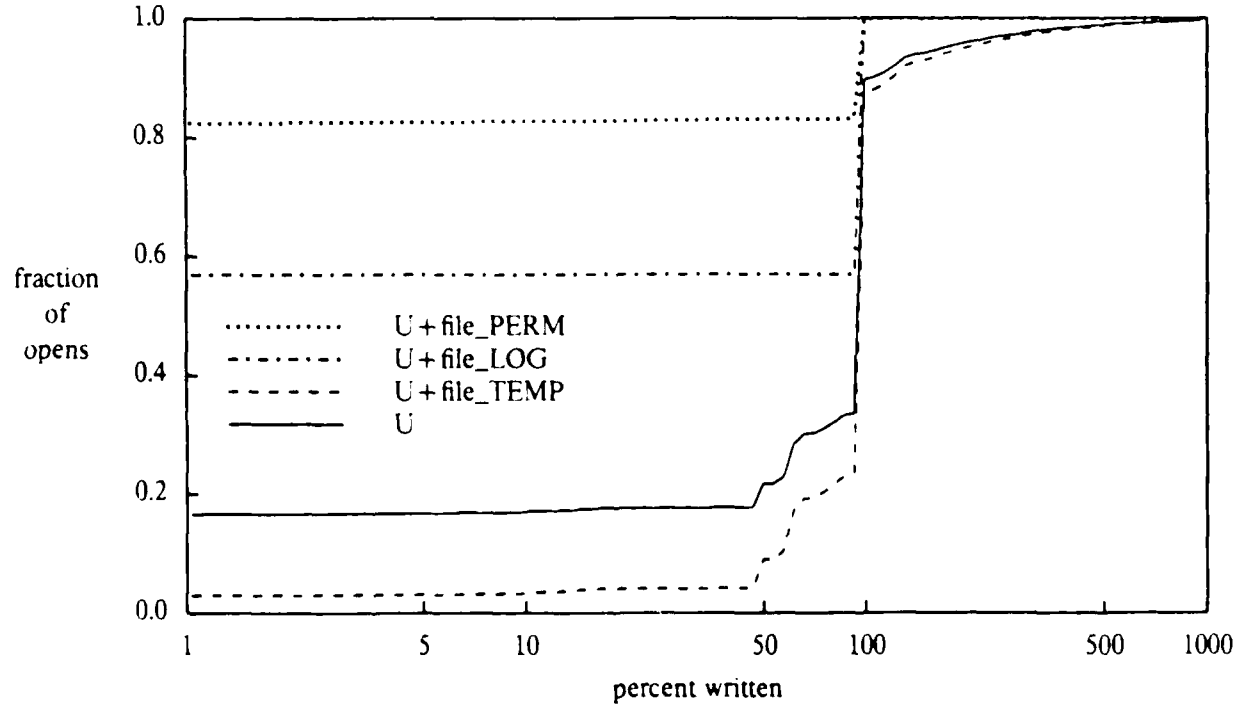


Figure 30: Percent of file written for read/write opens (cumulative, U cut)

distribution	min	max	mean	median	std dev	<100%	>100%
U+file_LOG	0	100	75.3	100	40	28%	0%
U+file_PERM	0	12500	99.9	100	110	5.7%	5.2%
U+file_TEMP	0	1160	109	100	47	7.2%	11.4%
U	0	12500	100.9	100	104	5.9%	6.0%
no cut (table 11)	0	64100	83.9	100	202	31%	2.9%

Table 35: Percentage of users files read (read-only opens)

distribution	min	max	mean	median	std dev	<100%	>100%
U+file_LOG	0	100	12.4	1.9	26	94%	0%
U+file_PERM	0	200	90.8	100	27	14%	0.8%
U+file_TEMP	0	9600	106.4	100	201	2.0%	0.7%
U	0	9600	95.6	100	134	11%	0.8%
no cut (table 12)	0	9600	85.7	100	53	15%	0.1%

Table 36: Percentage of user files written (write-only opens)

distribution	min	max	mean	median	std dev	<100%	>100%
U+file_LOG	0	100	60.2	100	49	40%	0%
U+file_PERM	0	100	81.8	100	39	18%	0%
U+file_TEMP	0	65000	159	100	1670	34%	19%
U	0	65000	145	100	1500	31%	16%
no cut (table 13)	0	65000	82.9	100	615	40%	11%

Table 37: Percentage of user files read (read/write opens)

distribution	min	max	mean	median	std dev	<100%	>100%
U+file_LOG	0	100	43.0	<1	50	57%	0%
U+file_PERM	0	100	17.0	<1	37	83%	1%
U+file_TEMP	0	3600	112	100	156	23%	12%
U	0	3600	95.7	100	147	34%	10.3%
no cut (table 14)	0	20000	51.8	<1	249	61%	2.7%

Table 38: Percentage of user files written (read/write opens)

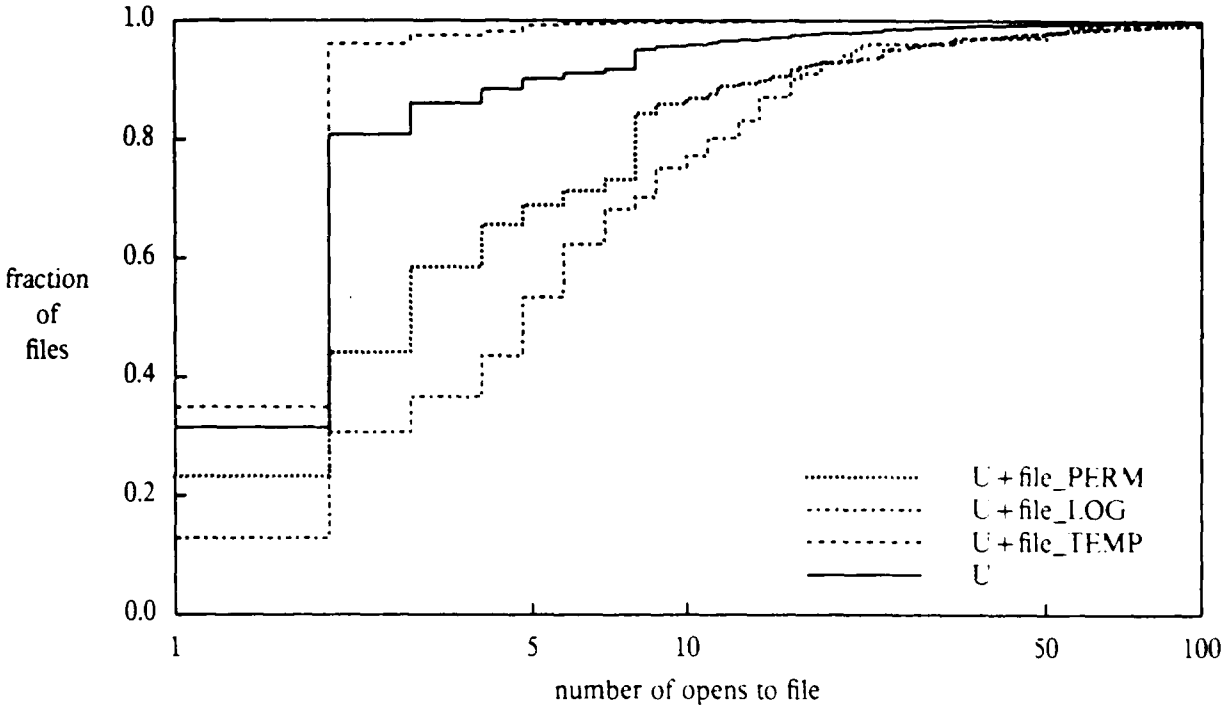


Figure 31: Number of opens per active file (cumulative, U cut)

distribution	mean	median	opened once	opened twice	opened more than twice	max
U + file_LOG	8.3	5	13%	18%	69%	79
U + file_PERM	7.5	3	23%	21%	56%	562
U + file_TEMP	1.8	2	35%	61%	3.9%	198
U	3.5	2	31%	50%	19%	562
no cut (table 17)	7.5	2	48%	33%	19%	26800

Table 39: Number of user opens/user file

6.3.3. Per File Results for User Files

User temp files were generally accessed twice. User log and perm files saw somewhat more activity (figure 31 and table 39). Although only 19% of the user files seen were referenced more than twice during the week of logging, these files accounted for 63% of the opens. User file distributions don't show the frantic activity to a few files that we saw for the overall distribution, but there was still a small group of relatively active files that accounted for the majority of the opens.

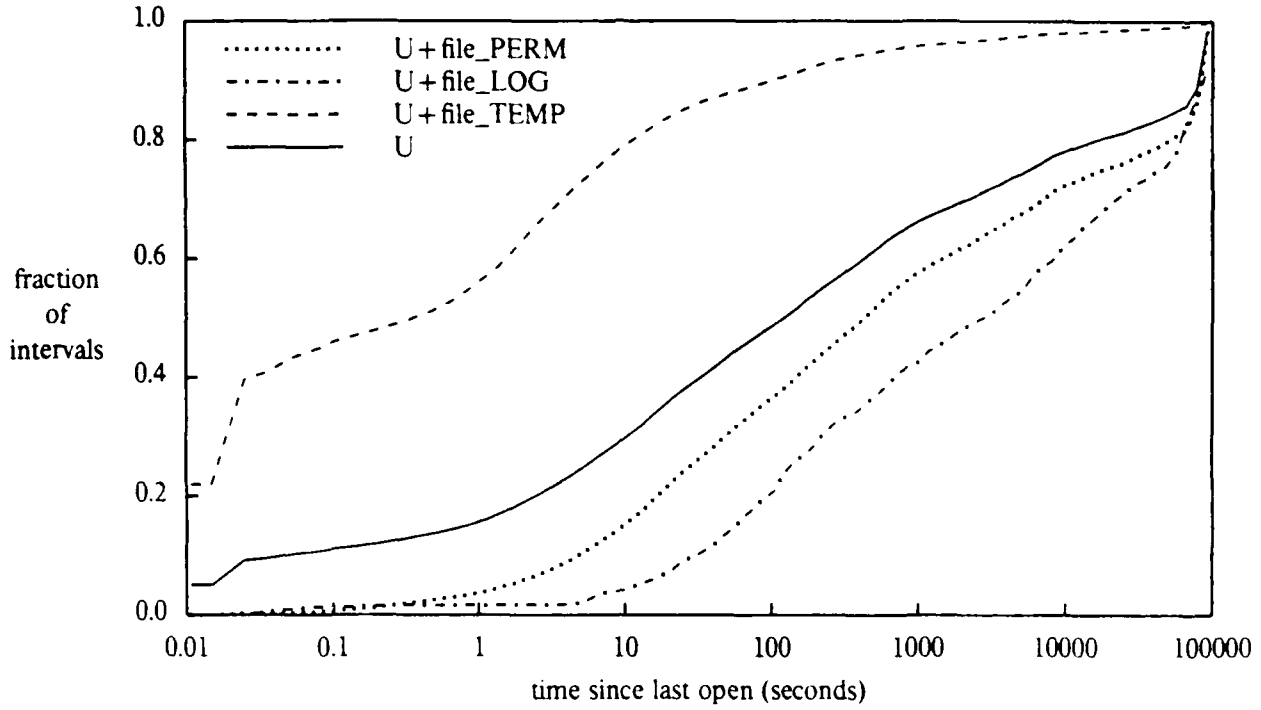


Figure 32: File interopen intervals (cumulative, U cut)

distribution	min	max	mean	median	std deviation
U+file_LOG	0.02	5.4e5	31100	3100	5.7e5
U+file_PERM	0	5.4e5	21400	450	4.1e4
U+file_TEMP	0.01	4.2e5	1390	0.38	1.2e4
U	0	5.4e5	16900	120	3.7e4
no cut (table 20)	0	5.4e5	7502	60	2.2e4

Table 40: User file interopen intervals (seconds)

Interopen intervals for user files (figure 32 and table 40) bore little resemblance to the results we saw for the overall data. Intervals for user files could ıerally be expressed in minutes instead of seconds. Temp files were an exception here. The second opc. to a temp file usually followed immediately after the first one.

File and data lifetimes for user files are shown in figures 33 and 34. Most user perm and log files had lives exceeding our logging period. Data in user log and perm files was also long lived (this was not the case for the overall data). Half of all user temp files lived less than 15 seconds.

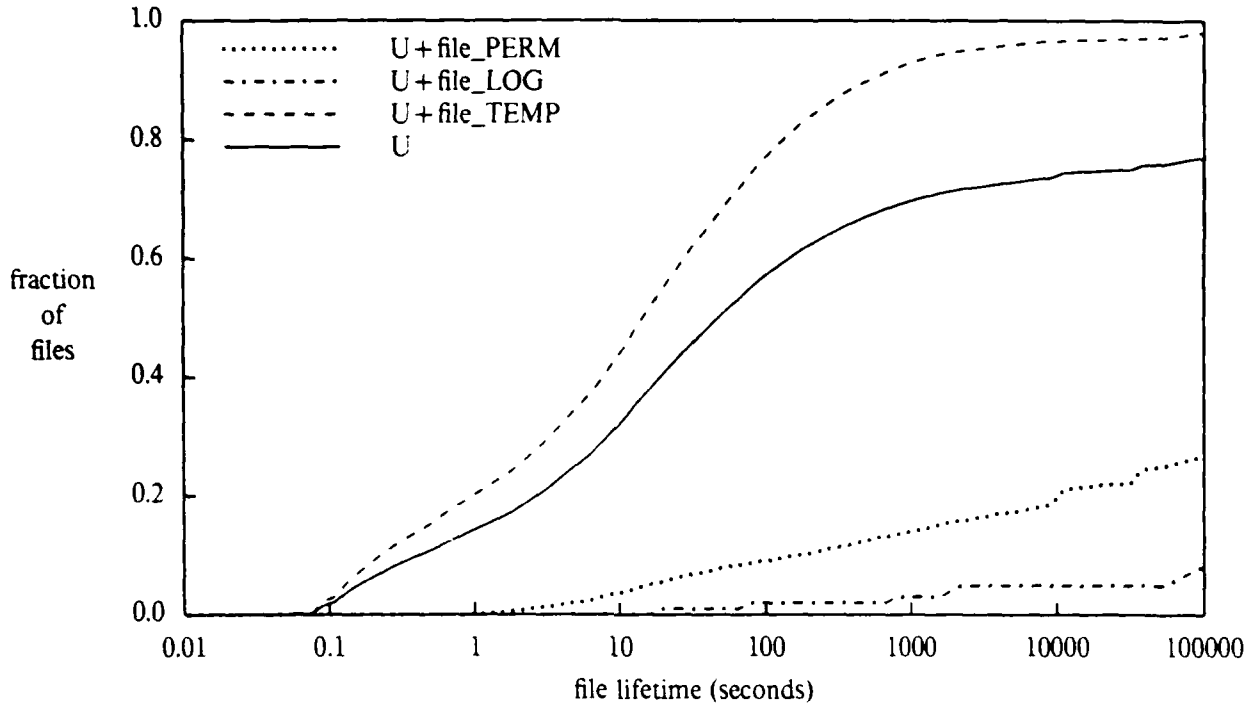


Figure 33: File lifetimes (cumulative, files living beyond log period binned at right, U cut)

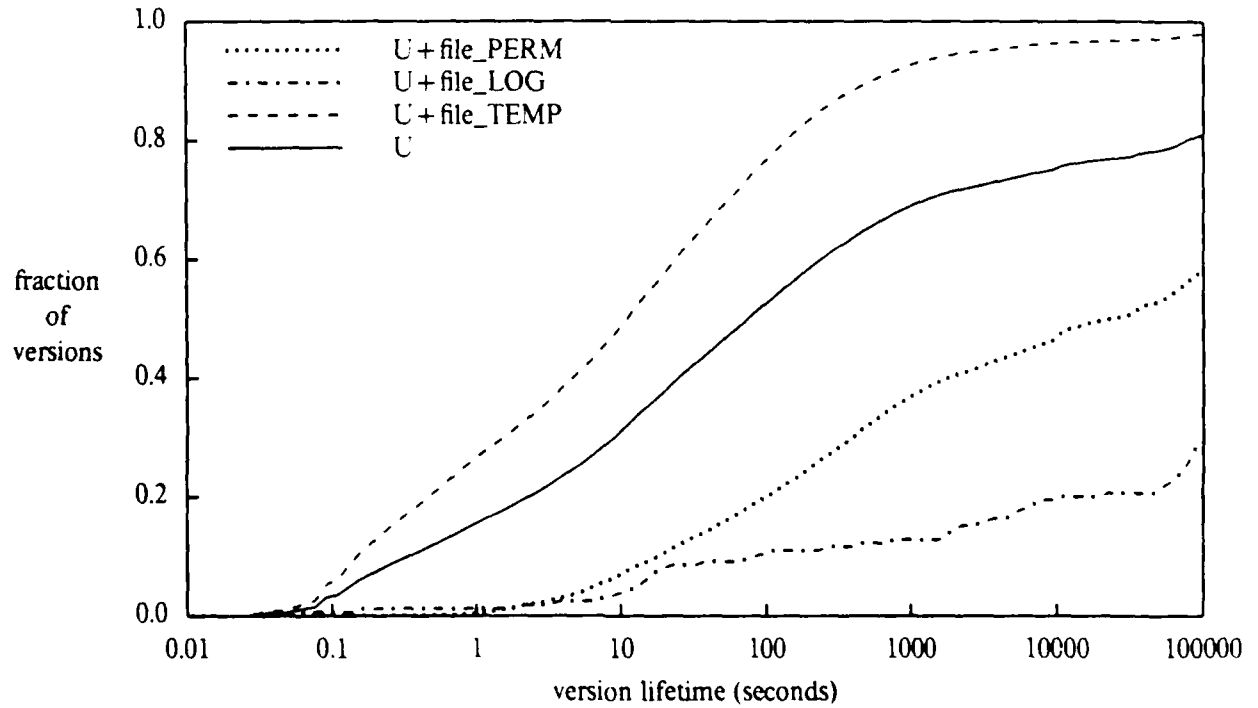


Figure 34: Version lifetimes (cumulative, versions living beyond log period binned at right, U cut)

Tables 41 and 42 provide some statistics on user sharing of user files. Sharing was restricted to log and perm files. The low mean number of inversions (0.069) indicates that sharing was incidental to the normal use of user files.

cut	readers		writers		users (r | w)		inversions	
	mean	>1	mean	>1	mean	>1	mean	max
U+file_LOG	0.634	3.0%	0.99	3.0%	1.099	5.9%	0.356	9
U+file_PERM	0.936	2.7%	0.632	2.1%	1.12	4.9%	0.231	163
U+file_TEMP	0.798	0.02%	0.995	0%	1.0	0.04%	0	2
U	0.838	0.80%	0.889	0.63%	1.035	1.5%	0.069	163
no cut (table 21)	0.792	6.6%	0.930	2.4%	1.30	11.8%	1.16	12529

Table 41: User file sharing

number	readers		writers		users (r | w)		inversions	
	count	cum	count	cum	count	cum	count	cum
0	5367	18.0%	3826	12.8%	-	-	29452	98.5%
1	24284	99.20%	25877	99.37%	29452	98.5%	198	99.20%
2	146	99.69%	98	99.70%	255	99.39%	109	99.56%
3	45	99.84%	28	99.80%	74	99.64%	27	99.65%
4	21	99.91%	19	99.86%	40	99.77%	23	99.73%
5	11	99.95%	8	99.89%	19	99.83%	14	99.78%
6	2	99.95%	7	99.91%	9	99.86%	7	99.80%
7	3	99.96%	7	99.93%	10	99.90%	5	99.82%
8	2	99.97%	3	99.94%	5	99.91%	7	99.84%
9	2	99.98%	5	99.96%	7	99.94%	7	99.86%
10	0	99.98	2	99.97%	2	99.94%	4	99.88%
>10	7	100%	10	100%	17	100%	37	100%
total	29890	-	29890	-	29890	-	29890	-

Table 42: readers, writers, users and inversions; user references to user files

7. Implications for DFS's

In this section we make some observations on DFS design, based on the results we have presented. It should be emphasized that these observations and suggestions are most applicable to systems that see reference patterns similar to ours. They will not necessarily carry over to other environments.

The small median size of opened files (710 bytes) suggests that the overhead to traverse a directory and then actually open a file will tend to dominate file access time. Careful directory design and low communication requirements for opens will be needed to minimize this overhead.

The high percentage of a file that was read or written tells us that migrating a file as a whole is usually appropriate. Log files are an exception[7]. In this case, information about the intended use of the file would be helpful.

For files that were not completely read or written, the fraction accessed depended strongly on the access mode (read-only, write-only or read/write), the size of the file and the opener of the file. Systems limited by bandwidth considerations may benefit from using this information in making migration decisions.

Reads accounted for 84% of the bytes transferred in the system. Many of these reads were from large administrative files that were frequently read and rarely written. Replication and caching of even a few such files could substantially increase the performance of a DFS.

Most temp files in our environment were opened only once or twice. These files were also short lived, generally existing for only a few seconds. Many other files were only used a few times during the logging period. Knowing the intended use of these files at the time of their creation could substantially increase the performance of a DFS. There is, for example, no need to replicate a temp file and files that are infrequently used or short-lived will usually benefit from different initial placement decisions.

The short interopen intervals seen here (median of 60 seconds) suggests that fast response to changing patterns is important. DFS's that migrate or replicate a file at open time are often doing the right thing. User files had substantially longer interopen intervals. In some situations, fast response time will be less important for these files.

The bursty nature of requests (for our background activity in particular) means that congestion could be a serious problem at times. Preliminary results from the VICE/Andrew system [Svobodova 85] confirm the importance of this issue. It will be interesting to investigate algorithms that place and migrate files to minimize congestion.

User files accounted for only 15% of the open activity on our system. For DFS's that support access to local file systems coupled with access to a global user file system, minimizing the performance impact of adding the global file system on local accesses is clearly important.

Net and system files made up the bulk of shared files. Relatively few user files were shared and this sharing was incidental to their normal use. Overall, only about 12% of the files on the system were shared. This suggests that there is no need for replication to improve performance for most files (replication for increasing availability is another issue, though).

---

[7] One might prefer to use a different logging mechanism in a distributed environment in any case

Over half of all execute requests made went to just 13 files. These files occupied 0.46MB of disk space (0.08% of the total). This suggests that even a very modest amount of caching or other special treatment for such files will produce significant improvements in system performance.

There were generally substantial differences in access patterns for log, permanent and temporary files. Placement and migration algorithms will benefit from recognizing and ruthlessly exploiting these differences.

8. Further Work

The analysis of file system traces can soak up boundless amounts of time and energy. We have tried to stop at the point were we felt that we had enough information to understand trace driven simulations based on the data. There is a great deal of further work that could be done. Some possibilities include:

(1) Studies of open frequency as a function of file age. Smith found that for long term file reference patterns, open frequency falls off as the age of the file increases [Smith 81]. A recent survey of files on Seneca showing that 66% of all user files (user log and perm files) hadn't been accessed in over one month [Friedberg 85] suggests that this is also true in our environment for at least some classes of files.

(2) Studies of interopen intervals as a function of file size. Porcar found that smaller files tend to have shorter interopen intervals [Porcar 82]. We don't expect this to be true for the overall activity in our system (because of the large heavily used administrative files), but it may be true for user files.

(3) Measuring the paging and inode access activity. It would be interesting to see what fraction of the file system bandwidth is devoted to each of these activities.

(4) Examining in more detail the activity per user. Ousterhout et al. [Ousterhout 85] have done some of this work.

(5) Using the trace data to drive simulations investigating file system performance issues. A trace driven simulation of Roe [Ellis 83] is planned.

(6) Fitting curves to various distributions (size, inter-open time and so on). These would be useful in writing synthetic drivers for use in simulating DFS's [Satyanarayanan 83].

(7) Further data collection and analysis for different environments and work loads. This would give us a better feeling for where our data fits into the universe of file system usage.

9. Summary

This paper has described in detail the collection and analysis of short term file reference data from a 4.2BSD UNIX system supporting university research. Our major findings:

(1) Opened files in our environment are small, with half being under 710 bytes long.

(2) The majority of bytes read come from larger files (greater than 20,000 bytes long).

(3) 68% of files opened with read access are completely read and 78% of files opened with write access are completely written. The percentage read and written depends strongly on the class of the file (log, perm or temp), the mode of open, the file opener and the size of the file. In

particular, log files are almost never completely written and users completely read 94% of files they open read-only.

(4) Temporary files are usually accessed only once or twice and most live for less than a minute. Log and permanent files live for much longer periods and see more open activity.

(5) Most opens go to files opened hundreds or thousands of times a week. Large administrative files account for a substantial fraction of this activity.

(6) Files are generally open for only a few tenths of a second.

(7) Interopen intervals in our environment are short. Half are under 60 seconds. The interopen interval depends strongly on the class (log, permanent or temporary) and owner of the file.

(8) Most sharing is restricted to system and net files in our environment. Sharing of user files is incidental to their normal use.

(9) Executed files are relatively large (half are over 20,000 bytes), heavily used and few in number.

(10) Half of all execute requests go to a very small number of executable files (13 files; 2.2% of the referenced executables).

(11) We see substantial differences in file access patterns based on the class of the file, the owner of the file and the class of the file opener. In particular, overall reference patterns do not match user file reference patterns and reference patterns for logs, permanent files and temporary files bear little resemblance to each other.

These results have a number of interesting implications for DFS design. These implications are discussed in section 7.

As is true with all studies of this sort, our results can be guaranteed to be valid only for our system at the time of data collection. Care should be taken in applying the results to other situations.

10. Acknowledgements

Carla Ellis and Stuart Friedberg made numerous suggestions on both the analysis and presentation. Their help is gratefully acknowledged. Lee Moore's efforts in maintaining and enhancing our press software [Kahrs 85] helped make the plots shown here possible. Liudvikas Bukys and Mike Dean are to be thanked (I think) for convincing me that kernel hacking wasn't really so bad. Finally, I would like to thank the 5 VAXen that struggled so long and hard to analyze this data.

References

[Ellis 83] Ellis, C. and Floyd, R., "The Roe File System," *Proceedings of the Third Symposium on Reliability in Distributed Software and Database Systems*, October 1983, 175-81.

[Floyd 85] Floyd, R. A., "Short Term File Reference Patterns in a UNIX Environment: Preliminary Results," Internal Note, Department of Computer Science, University Rochester, August 1985.

[Floyd 86a] Floyd, R. A., *Transparency in Distributed File Systems*, Ph.D. Dissertation, Department of Computer Science, University Rochester, December 1986. (in preparation).

[Floyd 86b] Floyd, R. A., "Directory Reference Patterns in a UNIX Environment," Technical Report 178, Department of Computer Science, University of Rochester, May 1986. (in preparation).

[Friedberg 85] Friedberg, S., private communication, July 1985.

[Kahrs 85] Kahrs, M. and Moore, L., "Adventures with Typesetter-Independent TROFF," Technical Report 159, Department of Computer Science, University of Rochester, June 1985.

[Lazowska 84] Lazowska, E., Zahorjan, J., Cheriton, D. and Zwaenepoel, W., "File Access Performance of Diskless Workstations," Technical Report 84-06-01, Department of Computer Science, University of Washington, June 1984.

[Nowitz 78] Nowitz, D. and Lesk, M., "A Dial-Up Network of UNIX Systems," in *The UNIX Programmer's Manual, Seventh Edition*, vol. 2, Bell Laboratories, August 1978.

[Ousterhout 85] Ousterhout, J., Da Costa, H., Harrison, D., Kunze, J., Kupfer, M. and Thompson, J., "A Trace Driven Analysis of the UNIX 4.2BSD File System," UCB/Computer Science Department 85/230, EECS Department, University of California, Berkeley, April 1985.

[Porcar 82] Porcar, J., "File Migration in Distributed Computer Systems," LBL-14763, Lawrence Berkeley Laboratory, July 1982.

[Ritchie 78] Ritchie, D. and Thompson, K., "The UNIX Time-Sharing System," *Bell System Technical Journal 57*:6, Part 2, July-August 1978, 1905-30.

[Satyanarayanan 81] Satyanarayanan, M., "A Study of File Sizes and Functional Lifetimes," *Operating Systems Review 15*:5, December 1981, 96-108.

[Satyanarayanan 83] Satyanarayanan, M., "A Methodology for Modelling Storage Systems and its Application to a Network File System," CMU-CS-83-109, Department of Computer Science, Carnegie-Mellon University, March 1983.

[Satyanarayanan 85] Satyanarayanan, M., Howard, J., Nichols, D., Sidebotham, R., Spector, A. and West, M., "The ITC Distributed File System: Principles and Design," *Operating Systems Review 19*:5, December 1985, 35-50. (SOSP 10).

[Smith 81] Smith, A., "Analysis of Long Term File Reference Patterns for Application to File Migration Algorithms," *IEEE Transactions on Software Engineering SE-7*:4, July 1981, 403-417.

[Smith 85] Smith, A., "Disk Cache-Miss Ratio Analysis and Design Considerations," *ACM Transactions on Computer Systems 3*:3, August 1985, 161-204.

[Stritter 77] Stritter, E., "File Migration," STAN-CS-77-592, Stanford University, March 1977.

[Tichy 84] Tichy, W. and Zuwang, R., "Towards a Distributed File System," *1984 USENIX Summer Conference Proceedings*, June 1984, 87-97.

[Walker 83] Walker, B., Popek, G., English, R., Kline, C. and Thiel, G., "The LOCUS Distributed Operating System," *Operating Systems Review 17*:5, December 1983, 49-70. (SOSP 9).

[Zhou 85] Zhou, S., Da Costa, H. and Smith, A., "A File System Tracing Package for Berkeley UNIX," UCB/Computer Science Department 85/235, EECS Department, University of California, Berkeley, May 1985.

Appendix A. Frequently Opened and Executed Files

The following two tables list the most frequently opened and executed files during the logging period. What are actually listed here are the most frequently accessed **inodes**, with the given name being the path used to first access the inode. For the most part, this distinction doesn't matter. There are a few inodes listed here, though, that are one of several versions of a heavily used system file. An example is /etc/passwd, which starts life as /etc/ptmp. Occurrences of this are noted in the table.

rank	opens	fraction	path of first open
1	26801	5.4%	/etc/hosts
2	20675	4.1%	/usr/spool/rwho/whod.keuka
...	...	...	[2 more rwho daemon files]
5	14977	3.0%	/etc/passwd [35485 (7.1%) with /etc/ptmp versions]
6	12036	2.4%	/etc/utmp
7	10594	2.1%	/usr/spool/rwho/whod.capella
...	...	...	[9 more rwho daemon files]
17	9386	1.9%	/usr/include/whoami.h
18	8881	1.8%	/etc/ptmp [version of /etc/passwd]
19	8630	1.7%	/etc/ptmp [version of /etc/passwd]
20	7533	1.5%	/usr/lib/sendmail.st
21	7295	1.5%	/vmunix
22	6908	1.4%	/etc/termcap
23	6400	1.3%	/etc/group [6947 (1.4%) for all versions]
24	6294	1.3%	/etc/services
25	6211	1.2%	/etc/gettytab
26	5063	1.0%	/etc/ttys
27	3991	0.80%	/usr/lib/uucp/L.sys
28	3852	0.77%	/usr/lib/news/sys
29	3140	0.63%	/usr/lib/uucp/L.aliases
30	3111	0.62%	/usr/adm/lastlog
31	3039	0.61%	/etc/hosts.equiv
32	2765	0.55%	/usr/lib/news/nactive [9830 (2.0%) for all versions]
33	2613	0.52%	/bin/true
34	2303	0.46%	/usr/lib/uucp/SEQF
35	2295	0.46%	//.cshrc
36	2292	0.46%	/usr/lib/news/nactive
37	2247	0.45%	/usr/lib/aliases.dir
38	2246	0.45%	/usr/lib/aliases.pag
39	2165	0.43%	/usr/lib/news/nactive
40	2143	0.43%	/usr/lib/sendmail.cf

Table A-1: Frequently Opened Inodes

rank	executes	fraction	path of first execute
1	17519	14.0%	/bin/sh
2	6946	5.6%	/bin/rm
3	6511	5.2%	/bin/[
4	6402	5.1%	/bin/csh
5	4568	3.7%	/bin/ls
6	4497	3.6%	/etc/rdist
7	3776	3.0%	/usr/ucb/more
8	2517	2.0%	/usr/ucb/vi
9	2514	2.0%	/bin/login
10	2197	1.8%	/bin/echo
11	2152	1.7%	/usr/bin/rnews
12	2143	1.7%	/usr/lib/sendmail
13	2026	1.6%	/etc/logld
14	1891	1.5%	/bin/hostname
15	1803	1.4%	/bin/rmdir
16	1734	1.4%	/usr/ucb/mail
17	1667	1.3%	/usr/lib/uucp/uuxqt
18	1658	1.3%	/usr/lib/uucp/uucico
19	1612	1.3%	/bin/stty
20	1592	1.3%	/usr/ucb/tset
21	1520	1.2%	/bin/mail
22	1382	1.1%	/usr/bin/uux
23	1283	1.0%	/bin/cat
24	1260	1.0%	/bin/eftpsend
25	1169	0.93%	/etc/getty
26	1013	0.80%	/etc/dmesg
27	999	0.80%	/usr/lib/news/batch
28	978	0.78%	/usr/ucb/clear
29	949	0.76%	/bin/mkdir
30	937	0.75%	/bin/awk
31	907	0.73%	/usr/bin/uux
32	882	0.71%	/etc/getty
33	876	0.70%	/bin/rmail
34	814	0.65%	/lib/cpp
35	812	0.65%	/usr/bin/basename
36	791	0.63%	/usr/ucb/uptime
37	784	0.63%	/bin/cc
38	772	0.62%	/bin/date
39	745	0.60%	/usr/lib/news/compress
40	744	0.59%	/bin/as

Table A-2: Frequently Executed Inodes

Appendix B. Selected Histograms and Distributions, in Detail

The first two tables in this appendix (table B-1 and table B-2) give information on opens and bytes transferred for each of the 14 cuts described in section 4. Some of the information in these tables appeared in the main body of the paper and is included again here for comparison purposes.

The remainder of the appendix gives a more complete set of distributions for file sizes, percent read and written, open counts, open time, interopen intervals and lifetimes. Distributions for all of our cuts are given. Again, some of this information also appears in the body of the paper.

cut	opens	% opens	files	% files	opens/file
ruid_NET	249825	33.1%	51600	51%	4.8
ruid_SYSTEM	298186	39.5%	15500	15%	19.2
ruid_USER	206274	27.3%	45900	45%	4.5
owner_NET	249733	33.1%	46207	45.7%	5.4
owner_SYSTEM	392790	52.1%	25062	24.8%	15.7
owner_USER	111762	14.8%	30822	30.5%	3.6
file_LOG	35662	4.7%	506	0.5%	70.5
file_PERM	499193	66.2%	16352	16.2%	30.5
file_TEMP	219430	29.1%	84327	83.3%	2.6
U+file_LOG	837	0.1%	101	0.1%	8.3
U+file_PERM	65051	8.6%	8662	8.6%	7.5
U+file_TEMP	38420	5.1%	21127	20.9%	1.8
U	104308	13.8%	29890	29.5%	3.5
no cut	754285	100%	101185	100%	7.5

Table B-1: Opens to regular files

cut	reads		writes		overall (r+w)	
	bytes/sec	fraction	bytes/sec	fraction	bytes/sec	fraction
ruid_NET	870	21%	250	31%	1120	22.5%
ruid_SYSTEM	1060	25%	110	14%	1170	23.5%
ruid_USER	2260	54%	440	55%	2700	54%
owner_NET	845	20%	245	31%	1090	22%
owner_SYSTEM	2330	56%	130	16%	2460	49%
owner_USER	1015	24%	425	54%	1440	29%
file_LOG	45	1.1%	11	1.4%	56	1.1%
file_PERM	3225	77%	285	35%	3510	70%
file_TEMP	920	22%	505	63%	1425	29%
U+file_LOG	9.6	0.2%	4.6	0.6%	14.2	0.3%
U+file_PERM	400	10.0%	120	15%	520	10%
U+file_TEMP	570	14%	300	37%	870	17%
U	980	23%	420	53%	1400	28%
no cut	4190	100%	800	100%	4990	100%

Table B-2: Bytes read/written for regular files

B.2. Dynamic File Sizes

distribution	min	max	mean	median	std deviation
ruid_NET	0	9.48e5	16500	1230	5.4e4
ruid_SYSTEM	0	9.46e5	23200	178	7.5e4
ruid_USER	0	2.49e6	15300	1230	5.1e4
owner_NET	0	9.48e5	13700	1230	4.9e4
owner_SYSTEM	0	7.76e5	25600	310	7.4e4
owner_USER	0	2.49e6	6530	930	3.9e4
file_LOG	0	1.28e6	105000	38900	1.5e5
file_PERM	0	2.49e6	19600	620	5.9e4
file_TEMP	0	1.3e6	2980	620	1.9e4
U+file_LOG	0	1.28e6	90400	39000	1.7e5
U+file_PERM	0	2.49e6	6205	1230	4.0e4
U+file_TEMP	0	1.30e6	5006	310	2.4e4
U	0	2.49e6	6440	930	3.9e4
no cut	0	2.49e6	18800	710	6.2e4
static	0	7.95e6	8020	1600	5.6e4

Table B-3: File size distributions

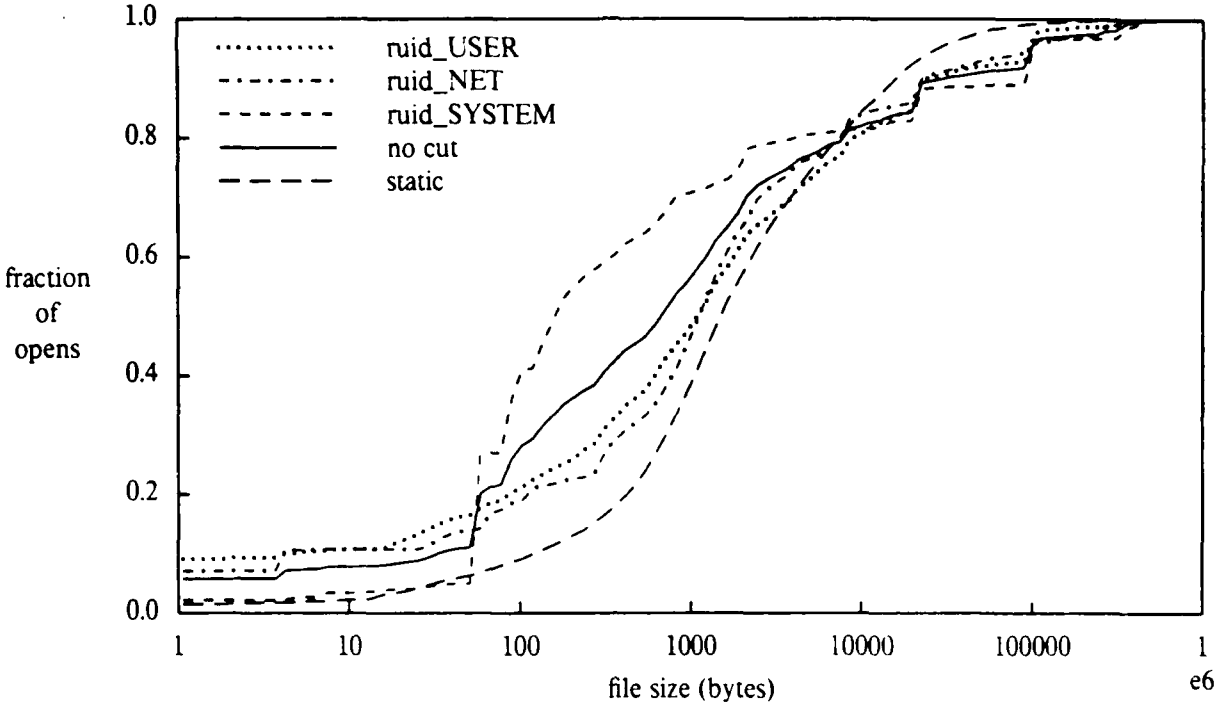


Figure B-1: Dynamic file size distributions (cumulative, measured at close, ruid cut)

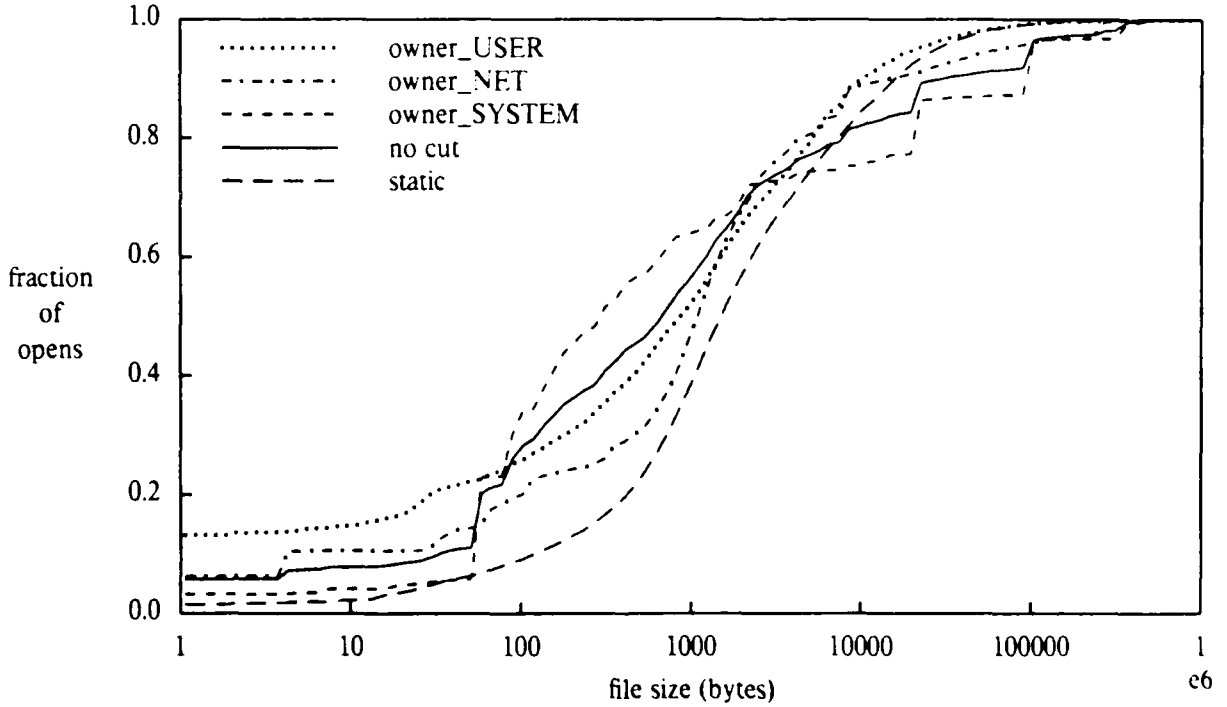


Figure B-2: Dynamic file size distributions (cumulative, measured at close, owner cut)

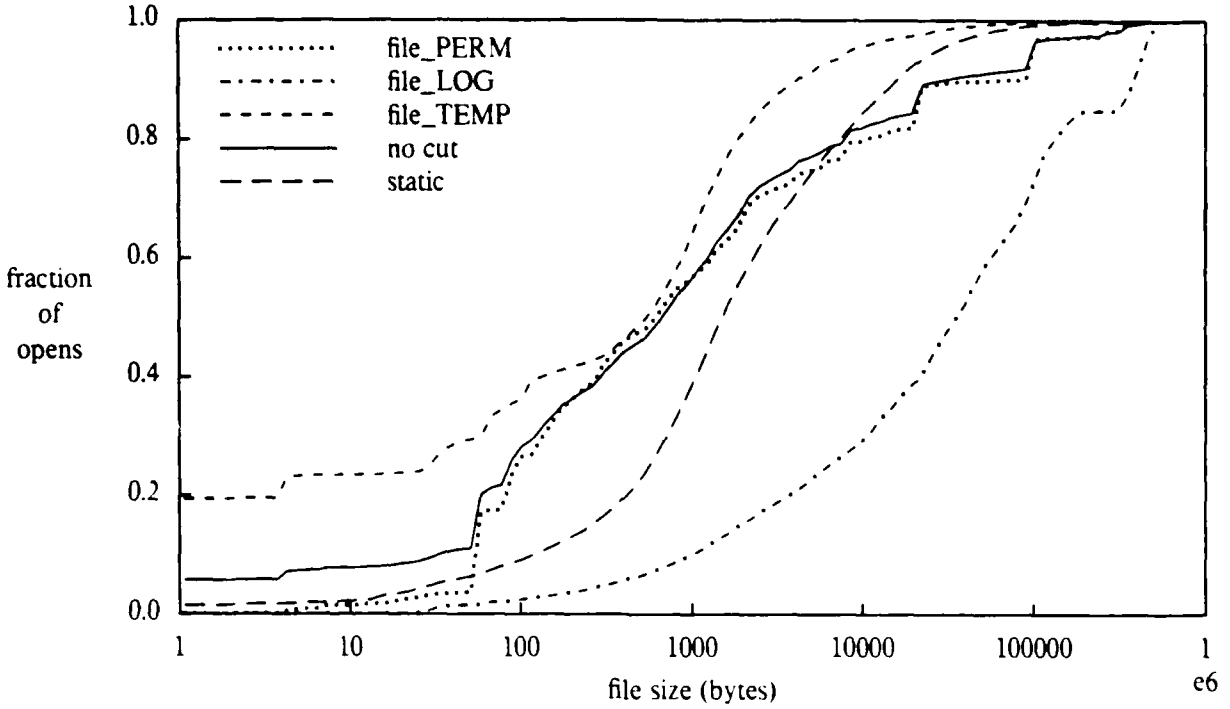


Figure B-3: Dynamic file size distributions (cumulative, measured at close, file cut)

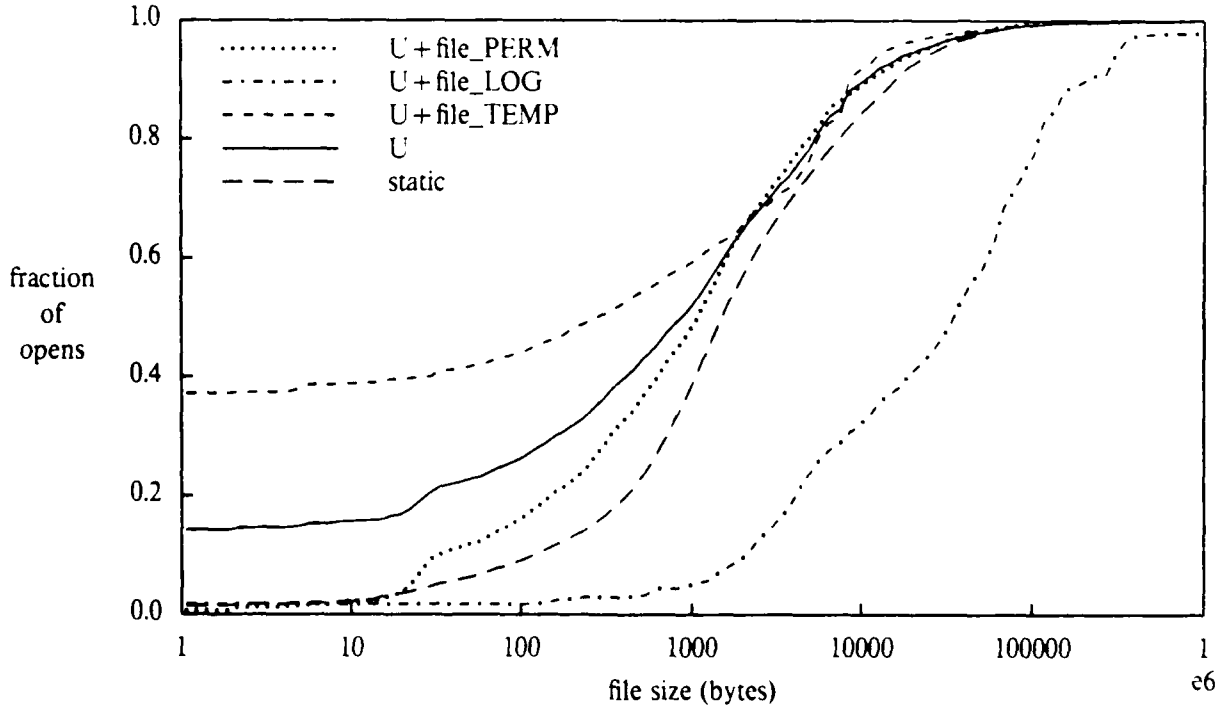


Figure B-4: Dynamic file size distributions (cumulative, measured at close, U cut)

B.3. Percentage Read (Read-Only Opens)

distribution	min	max	mean	median	std dev	<100%	>100%
ruid_NET	0	600	82.1	100	36	23%	0.3%
ruid_SYSTEM	0	3530	63.5	80	47	52%	3.0%
ruid_USER	0	64100	99.9	100	330	23%	5.5%
owner_NET	0	55500	99.0	100	264	16%	1.7%
owner_SYSTEM	0	64100	65.1	80	166	53%	3.0%
owner_USER	0	12500	100.4	100	99	6.1%	4.9%
file_LOG	0	690	85.8	100	72	26%	3.3%
file_PERM	0	64100	83.2	100	235	36%	3.2%
file_TEMP	0	3600	85.8	100	53	18%	2.1%
U+file_LOG	0	100	75.3	100	40	28%	0%
U+file_PERM	0	12500	99.9	100	110	5.7%	5.2%
U+file_TEMP	0	1160	109	100	47	7.2%	11.4%
U	0	12500	100.9	100	104	5.9%	6.0%
no cut	0	64100	83.9	100	202	31%	2.9%

Table B-4: Percentage read (read-only opens)

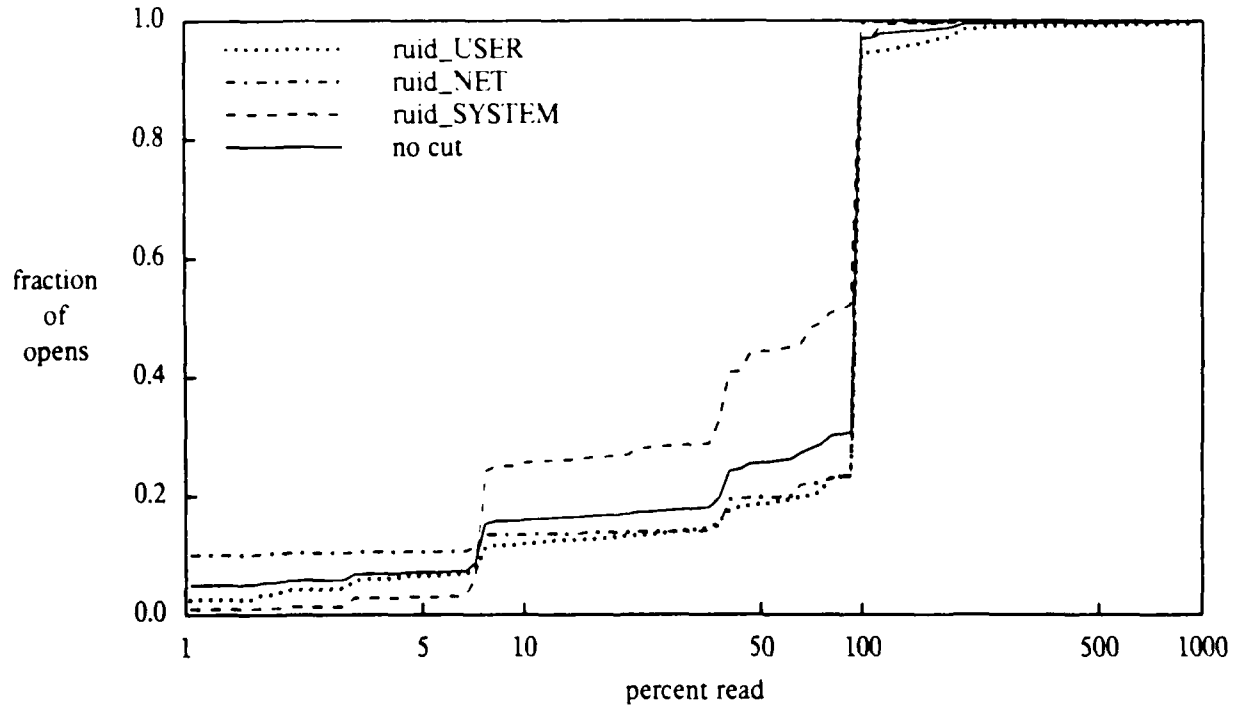


Figure B-5: Percent of file read for read-only opens (cumulative, ruid cut)

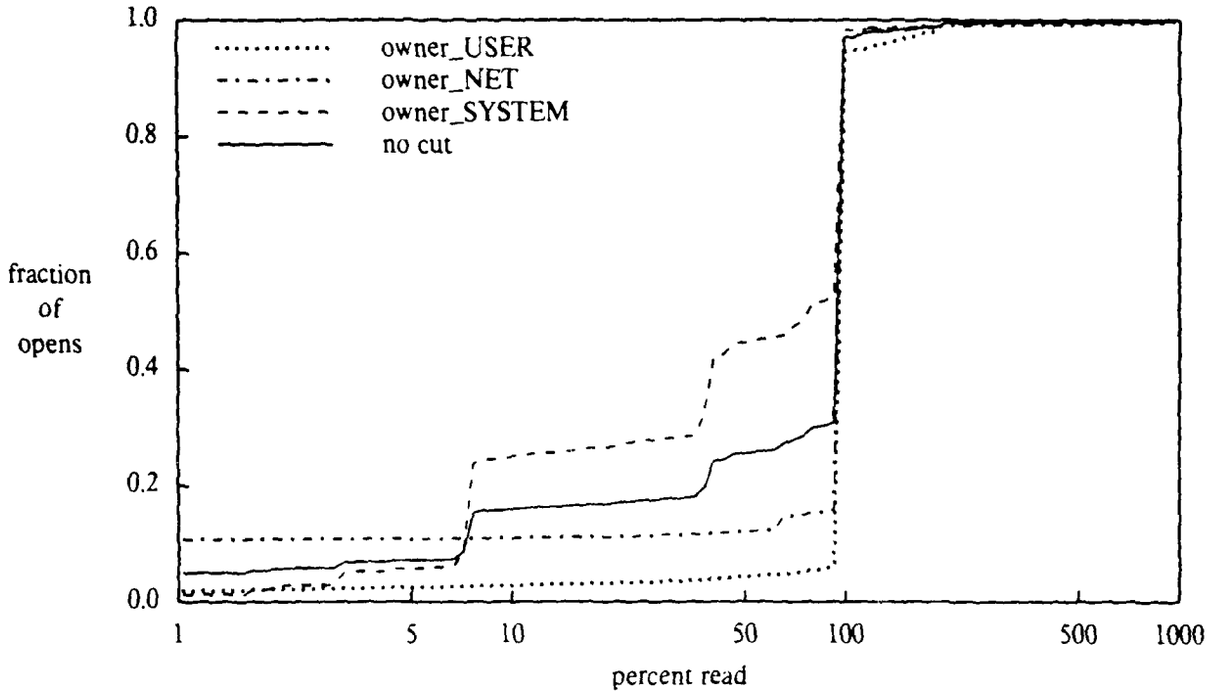


Figure B-6: Percent of file read for read-only opens (cumulative, owner cut)

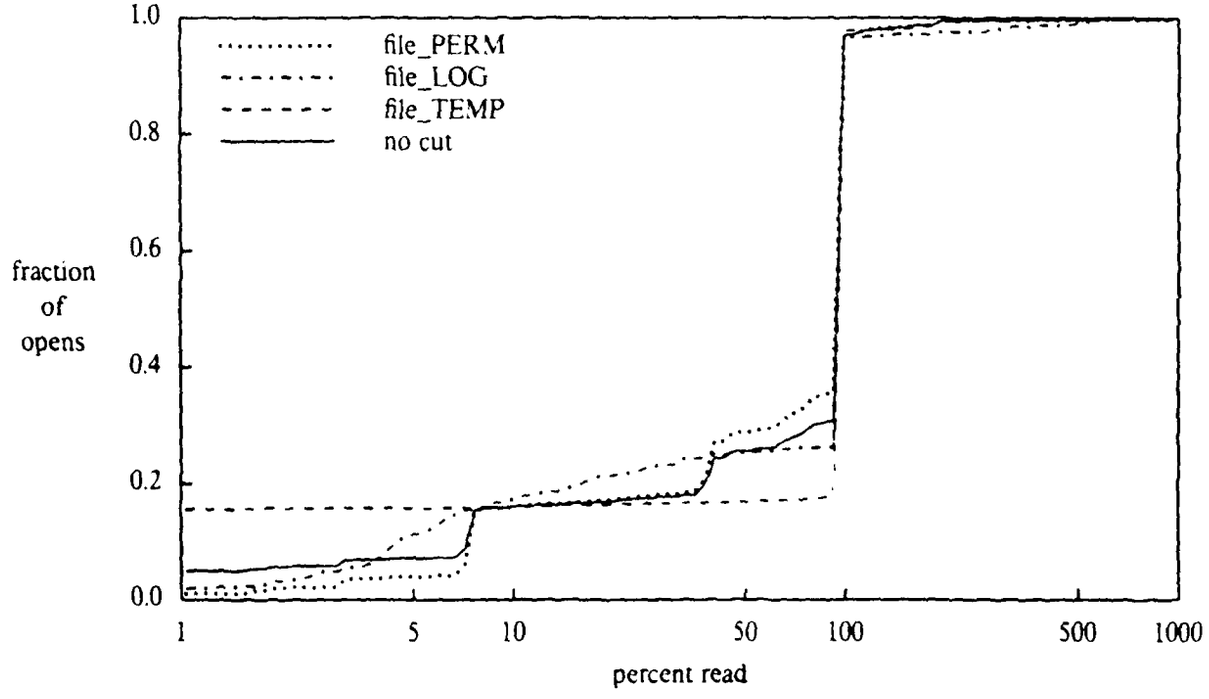


Figure B-7: Percent of file read for read-only opens (cumulative, file cut)

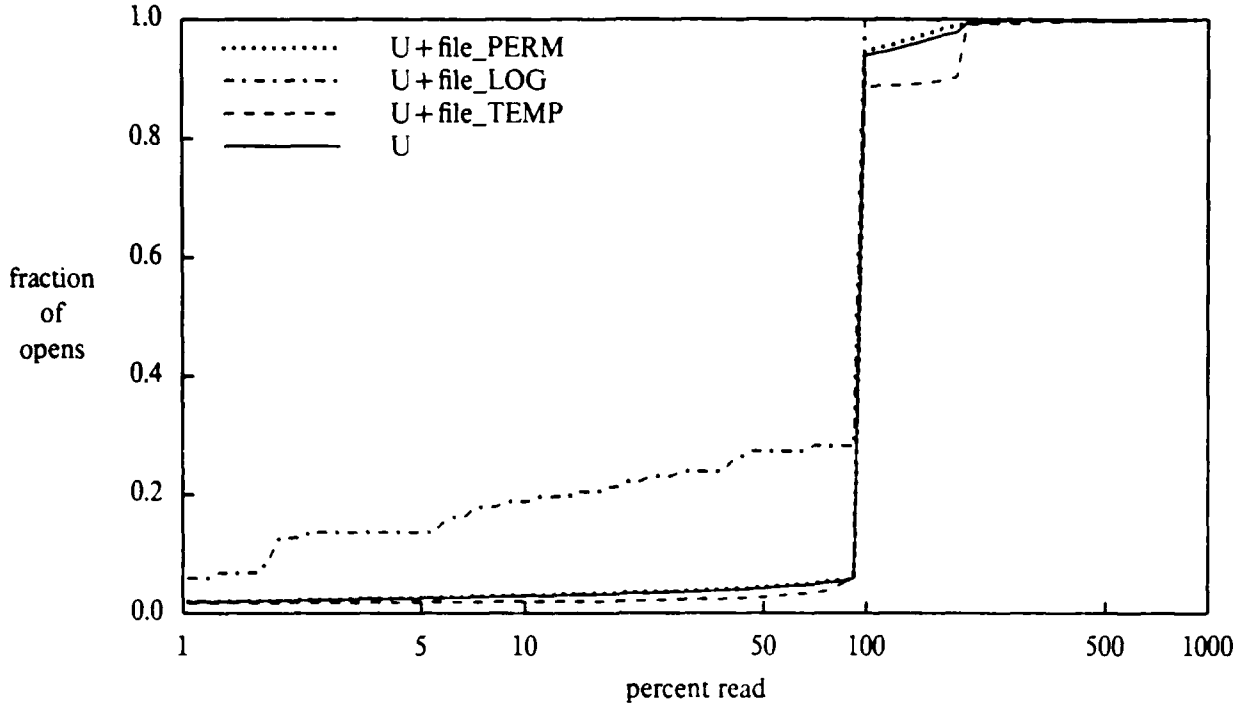


Figure B-8: Percent of file read for read-only opens (cumulative, U cut)

B.4. Percentage Written (Write-Only Opens)

distribution	min	max	mean	median	std dev	<100%	>100%
ruid_NET	0	100	60.2	100	48	41%	0%
ruid_SYSTEM	0	530	93.0	100	25	7.1%	0%
ruid_USER	0	9600	93.2	100	119	12%	0.6%
owner_NET	0	100	59.1	100	48	42%	0%
owner_SYSTEM	0	530	93.7	100	24	6.3%	0%
owner_USER	0	9600	93.7	100	132	11%	0.8%
file_LOG	0	100	2.8	<1	12	98.8%	0%
file_PERM	0	200	96.6	100	18	3.9%	0%
file_TEMP	0	9600	100.8	100	85	0.7%	0.2%
U+file_LOG	0	100	12.4	1.9	26	94%	0%
U+file_PERM	0	200	90.8	100	27	14%	0.8%
U+file_TEMP	0	9600	106.4	100	201	2.0%	0.7%
U	0	9600	95.6	100	134	11%	0.8%
no cut	0	9600	85.7	100	53	15%	0.1%

Table B-5: Percentage written (write-only opens)

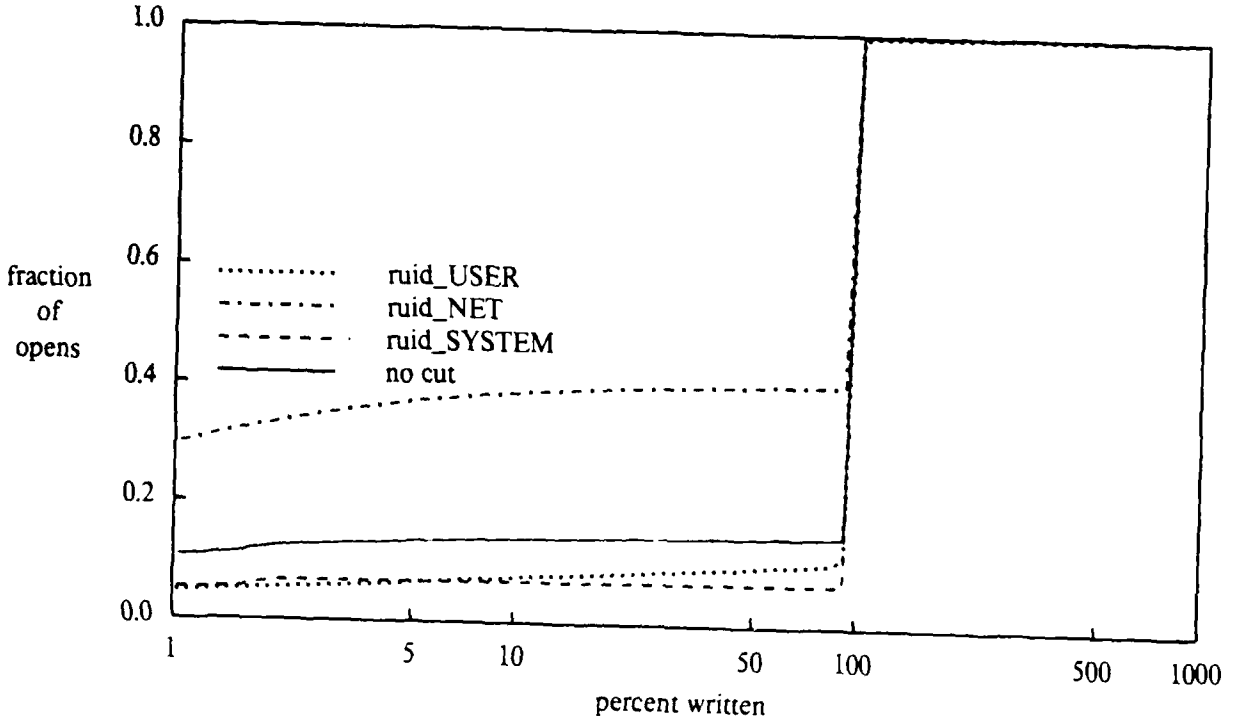


Figure B-9: Percent of file written for write-only opens (cumulative, ruid cut)

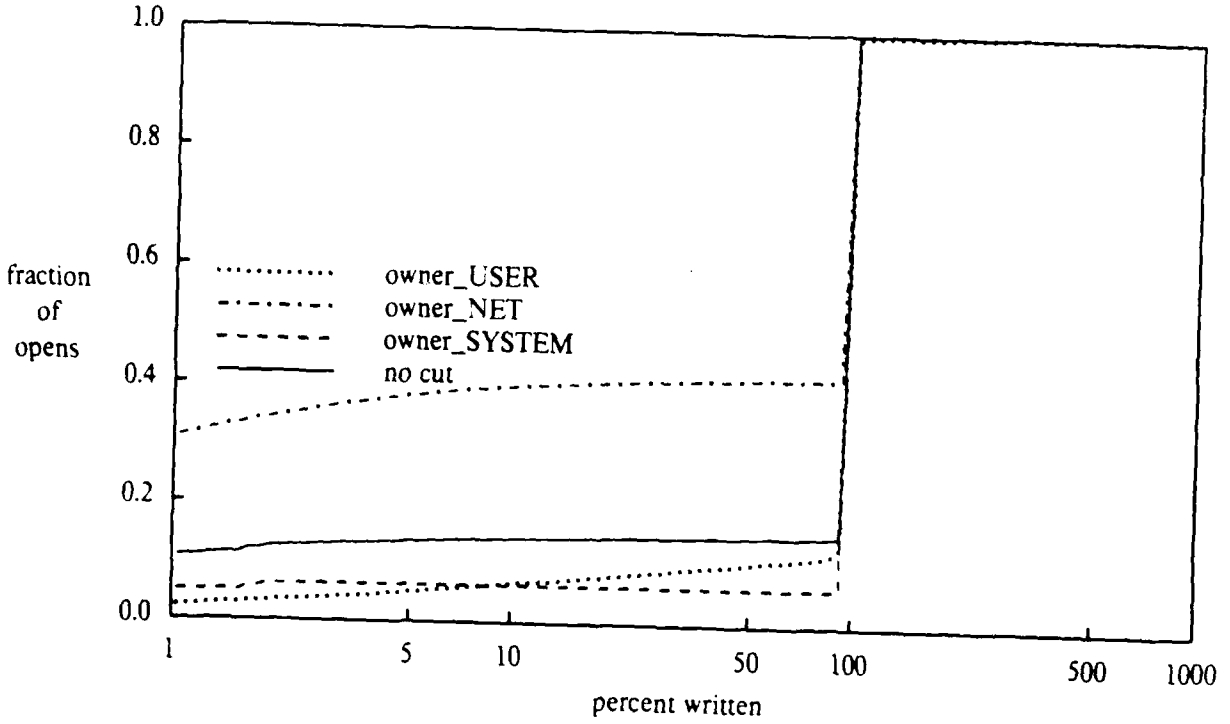


Figure B-10: Percent of file written for write-only opens (cumulative, owner cut)

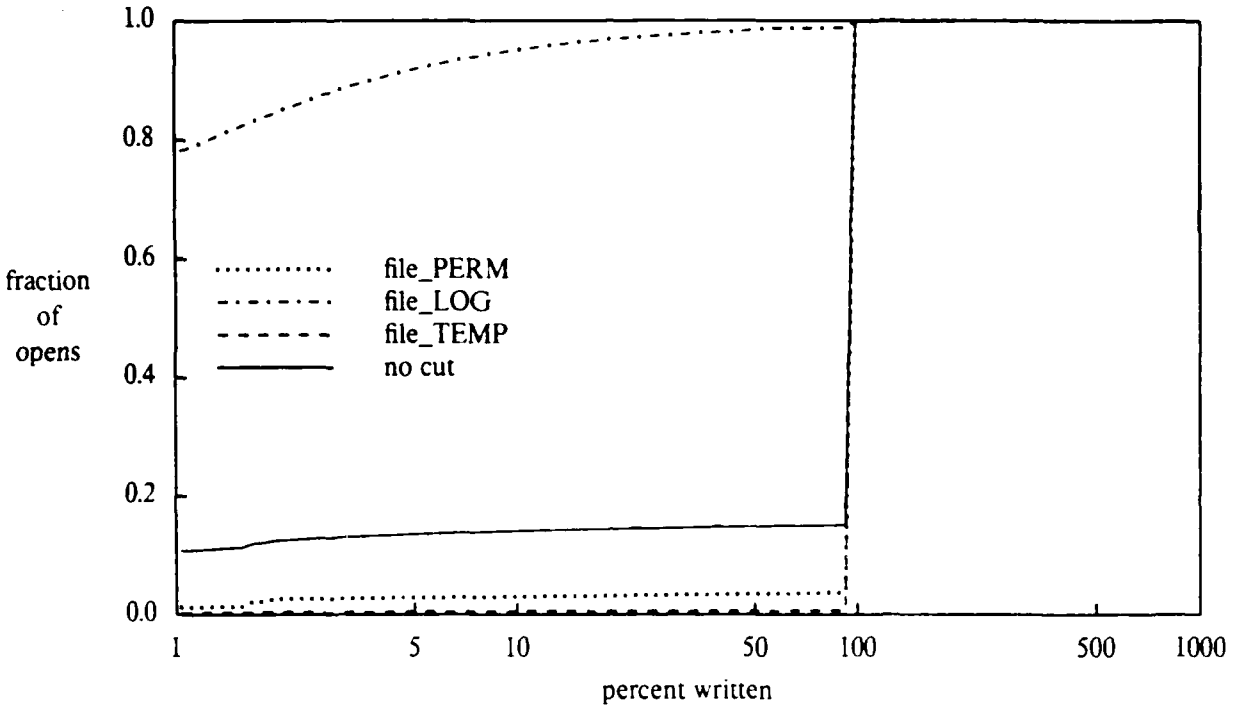


Figure B-11: Percent of file written for write-only opens (cumulative, file cut)

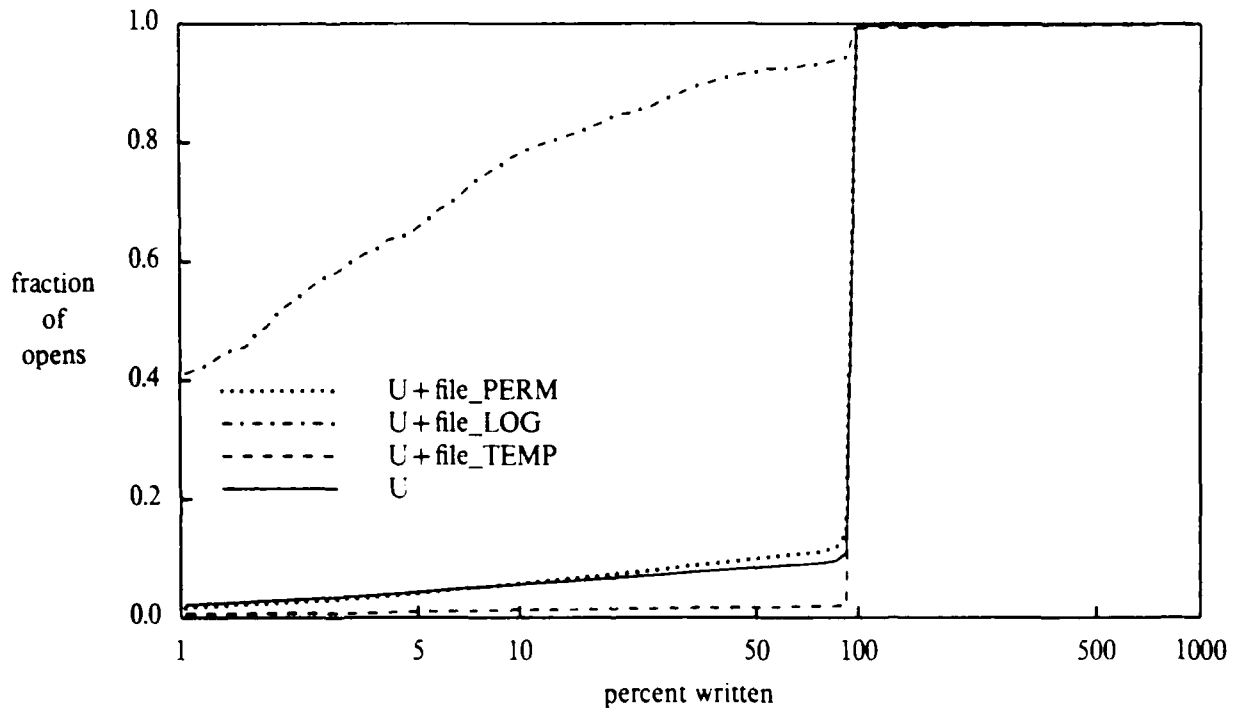


Figure B-12: Percent of file written for write-only opens (cumulative, U cut)

B.5. Percentage Read (Read-Write Opens)

distribution	min	max	mean	median	std dev	<100%	>100%
ruid_NET	0	2200	69.3	100	104	49%	15%
ruid_SYSTEM	0	1900	73.1	100	80	30%	0.2%
ruid_USER	0	65500	114	100	1100	33%	10%
owner_NET	0	2200	66.1	38	110	54%	18%
owner_SYSTEM	0	1900	76.6	100	65	27%	0.2%
owner_USER	0	65500	145	100	1500	31%	16%
file_LOG	0	100	60.2	100	49	40%	0%
file_PERM	0	1900	62.5	100	66	41%	0.4%
file_TEMP	0	65000	138	100	1180	37%	37%
U+file_LOG	0	100	60.2	100	49	40%	0%
U+file_PERM	0	100	81.8	100	39	18%	0%
U+file_TEMP	0	65000	159	100	1670	34%	19%
U	0	65000	145	100	1500	31%	16%
no cut	0	65000	82.9	100	615	40%	11%

Table B-6: Percentage read (read/write opens)

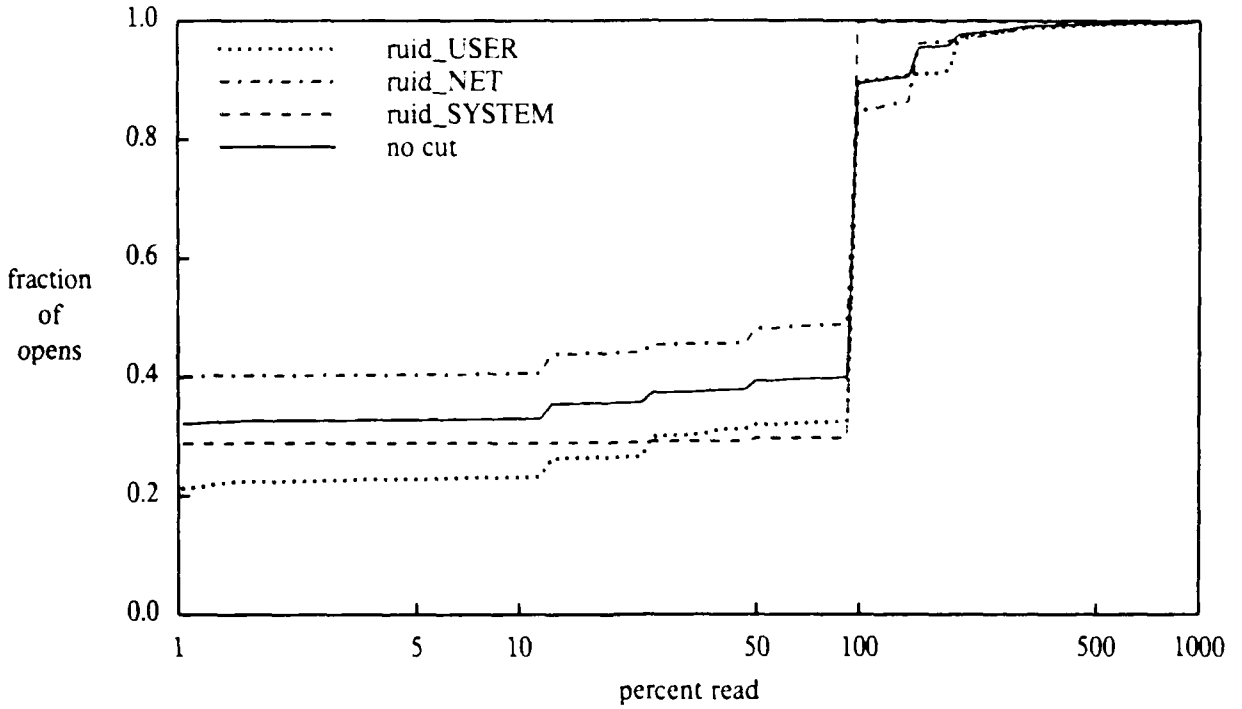


Figure B-13: Percent of file read for read/write opens (cumulative, ruid cut)

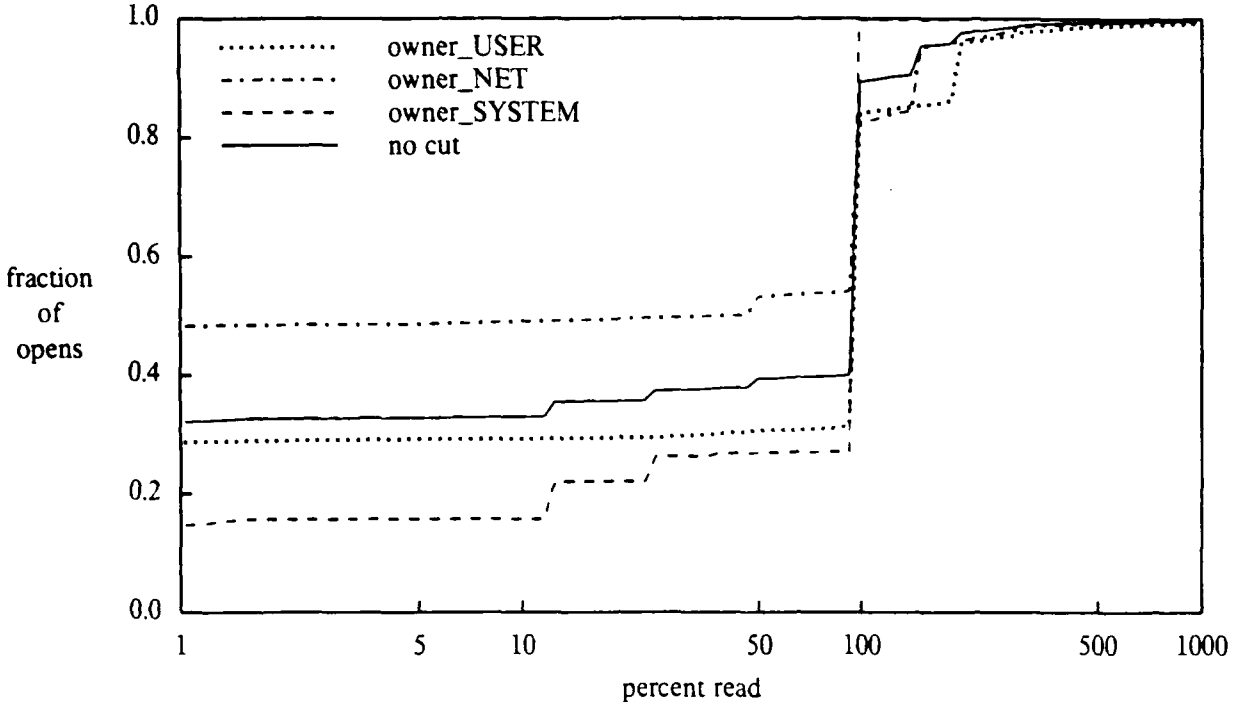


Figure B-14: Percent of file read for read/write opens (cumulative, owner cut)

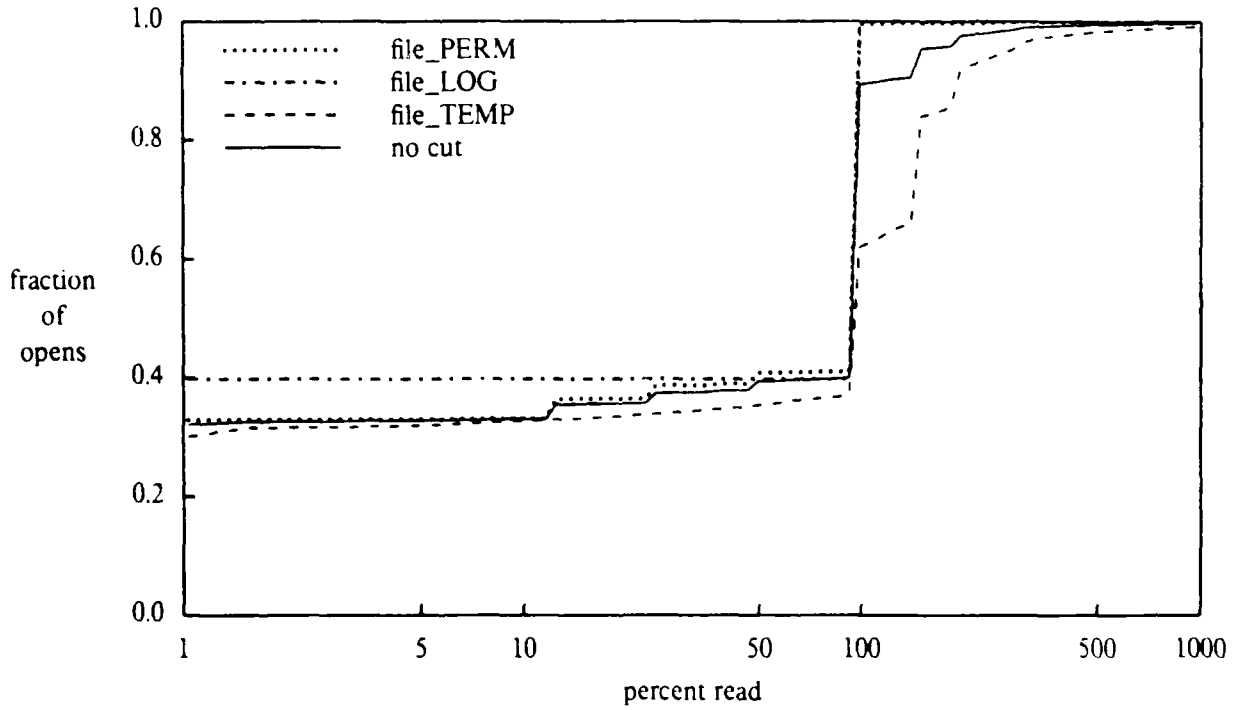


Figure B-15: Percent of file read for read/write opens (cumulative, file cut)

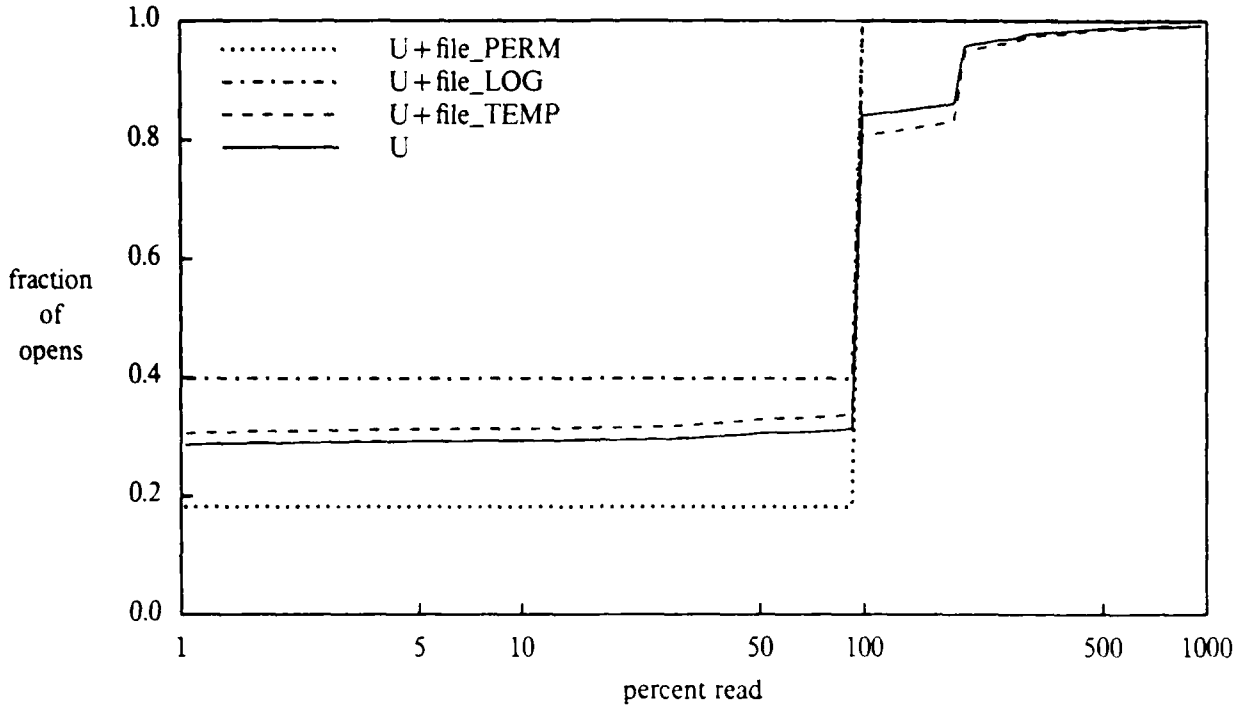


Figure B-16: Percent of file read for read/write opens (cumulative, U cut)

B.6. Percentage Written (Read-Write Opens)

distribution	min	max	mean	median	std dev	<100%	>100%
ruid_NET	0	20000	43.1	<1	326	70%	2.0%
ruid_SYSTEM	0	3600	43.9	1.7	159	64%	0.3%
ruid_USER	0	3600	72.8	100	118	44%	5.9%
owner_NET	0	20000	40.5	<1	340	74%	2.2%
owner_SYSTEM	0	3600	46.8	1.7	124	58%	0.2%
owner_USER	0	3600	95.7	100	147	34%	10%
file_LOG	0	100	43.0	<1	49.5	57%	0%
file_PERM	0	20000	36.4	<1	275	70%	0.1%
file_TEMP	0	3600	93.5	100	150	37%	9.8%
U+file_LOG	0	100	43.0	<1	50	57%	0%
U+file_PERM	0	100	17.0	<1	37	83%	1%
U+file_TEMP	0	3600	112	100	156	23%	12%
U	0	3600	95.7	100	147	34%	10.3%
no cut	0	20000	51.8	<1	249	61%	2.7%

Table B-7: Percentage written (read/write opens)

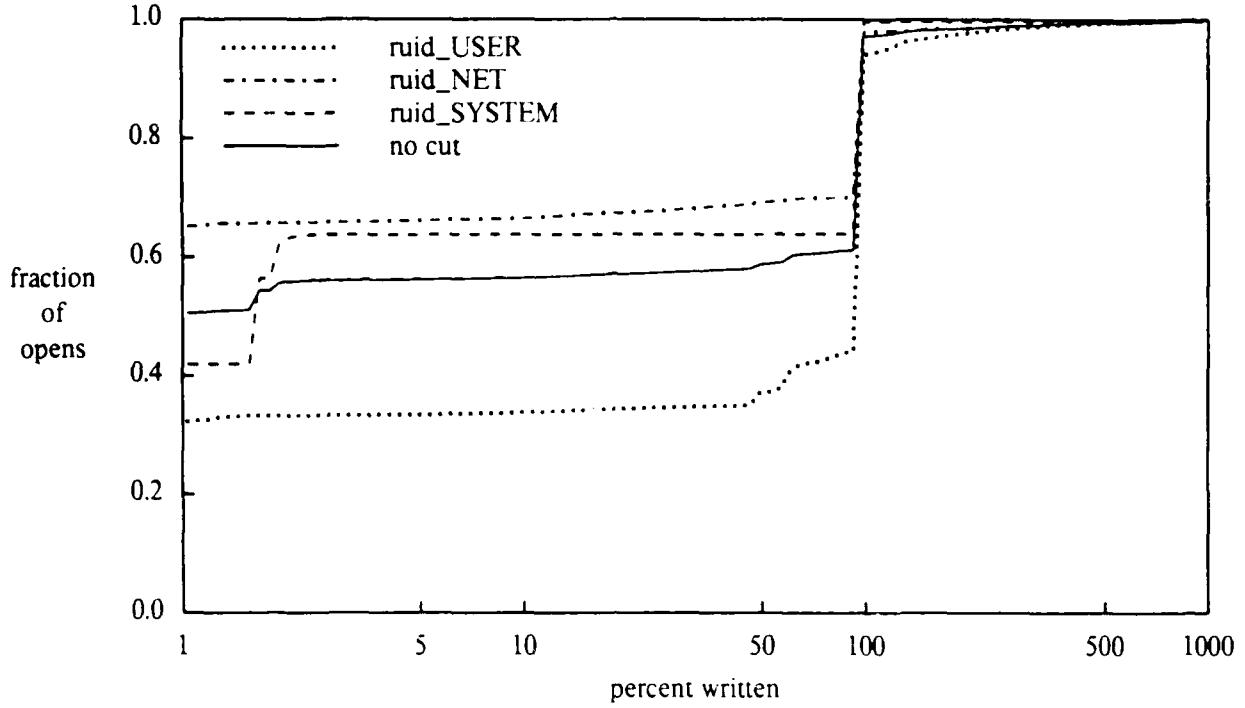


Figure B-17: Percent of file written for read/write opens (cumulative, ruid cut)

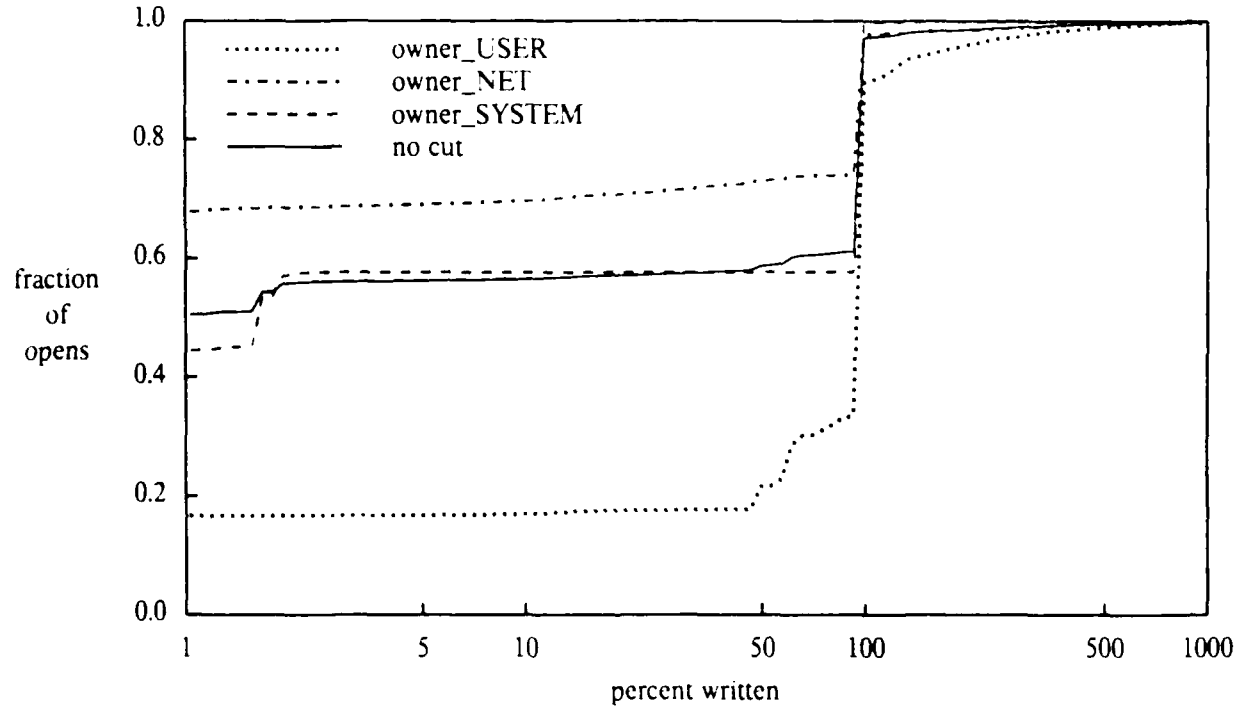


Figure B-18: Percent of file written for read/write opens (cumulative, owner cut)

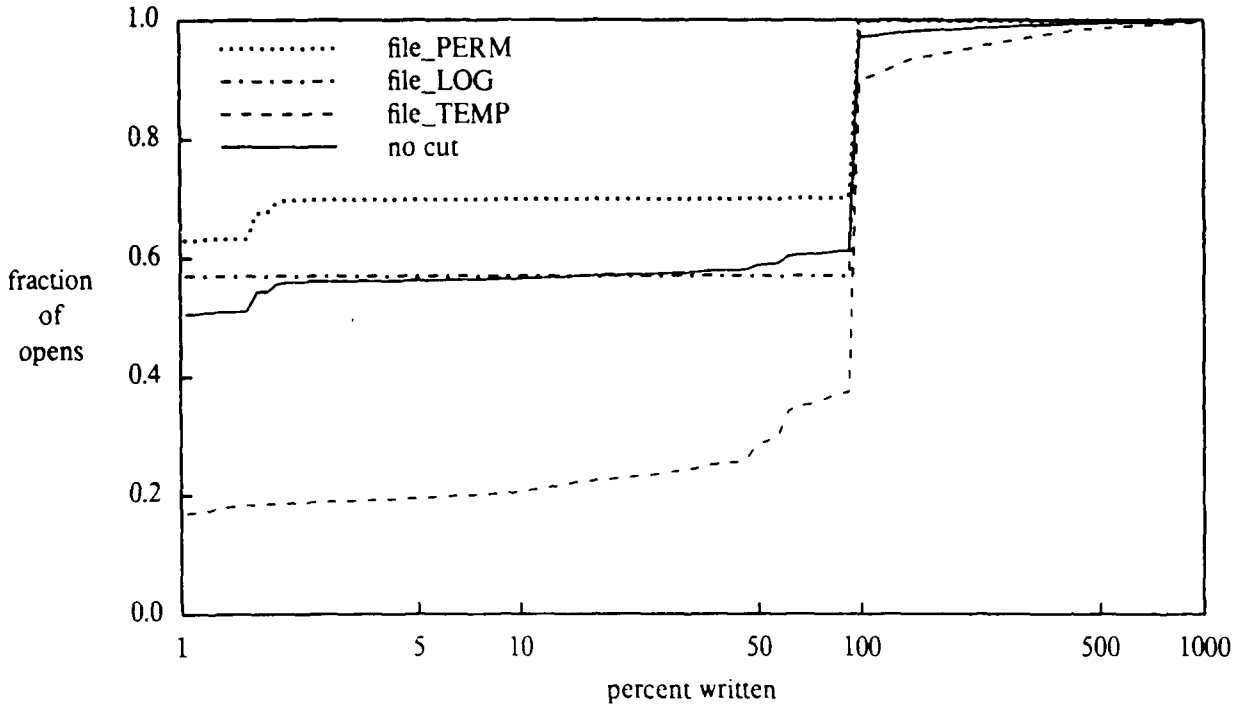


Figure B-19: Percent of file written for read/write opens (cumulative, file cut)

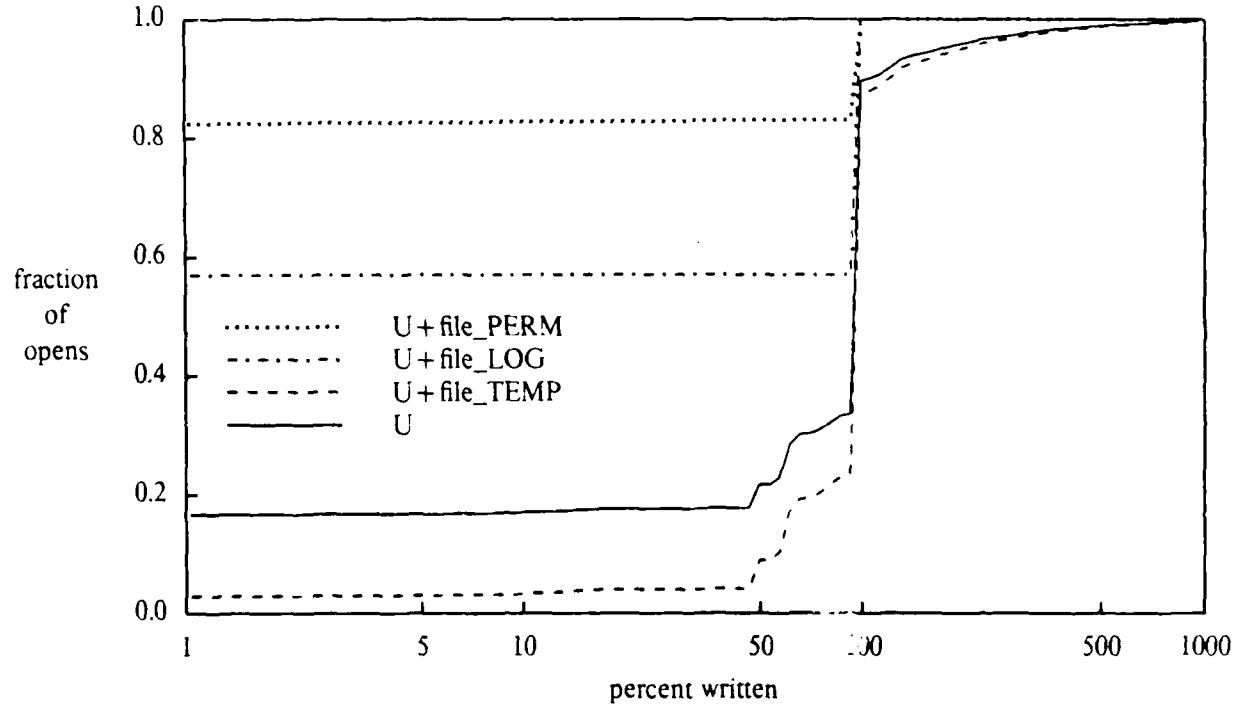


Figure B-20: Percent of file written for read/write opens (cumulative, U cut)

B.7. Number of Opens per File

distribution	mean	median	opened once	opened twice	opened more than twice	max
ruid_NET	4.8	1	56%	24%	20%	6140
ruid_SYSTEM	19.2	1	64%	22%	14%	20200
ruid_USER	4.5	2	39%	42%	19%	5390
owner_NET	5.4	1	51%	26%	23%	3990
owner_SYSTEM	15.7	1	67%	20%	10%	26800
owner_USER	3.6	2	32%	48%	20%	760
file_LOG	70.5	3	5%	36%	59%	5330
file_PERM	30.5	4	16%	16%	68%	26800
file_TEMP	2.6	1	55%	36%	9%	1920
U+file_LOG	8.3	5	13%	18%	69%	79
U+file_PERM	7.5	3	23%	21%	56%	562
U+file_TEMP	1.8	2	35%	61%	3.9%	198
U	3.5	2	31%	50%	19%	562
no cut	7.5	2	48%	33%	19%	26800

Table B-8: Number of opens/file

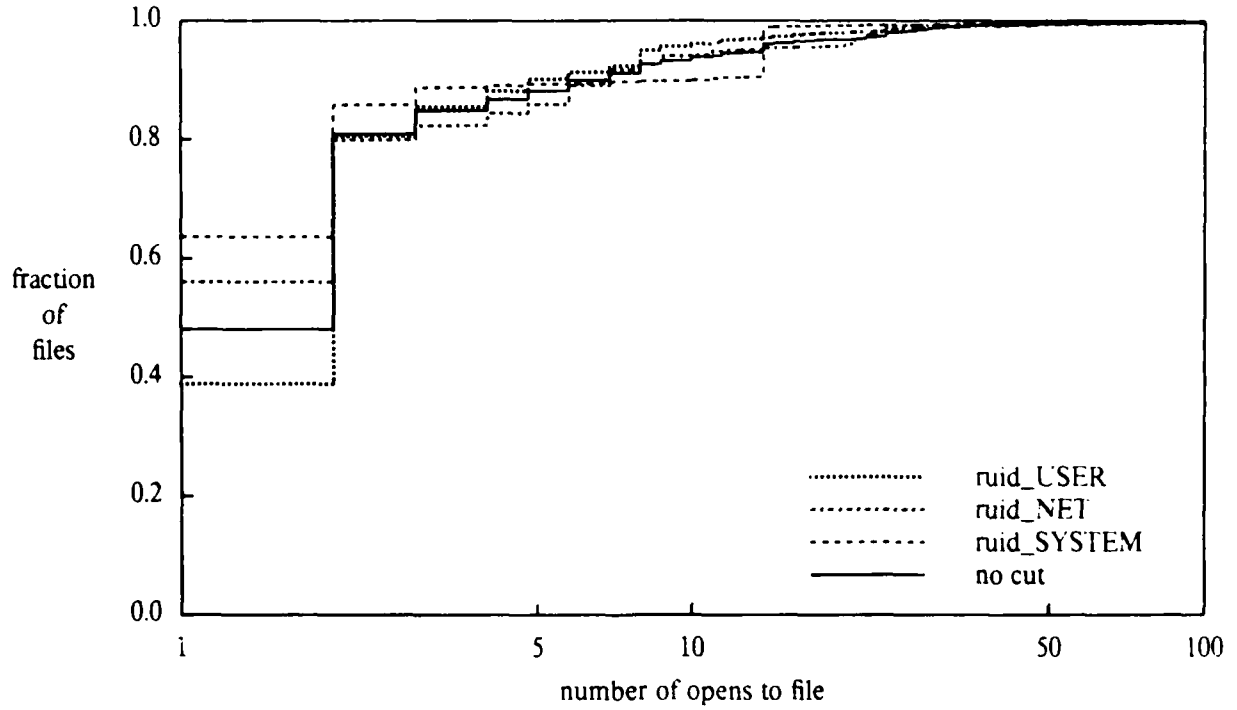


Figure B-21: Number of opens per active file (cumulative, ruid cut)

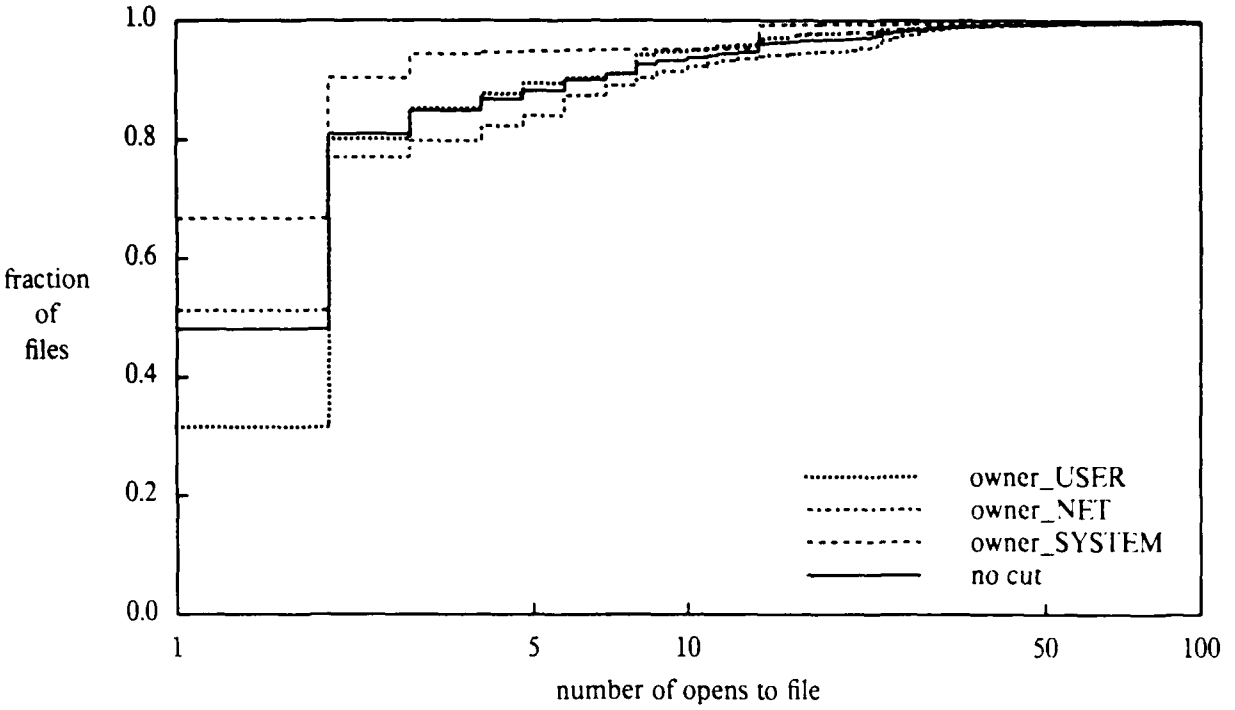


Figure B-22: Number of opens per active file (cumulative, owner cut)

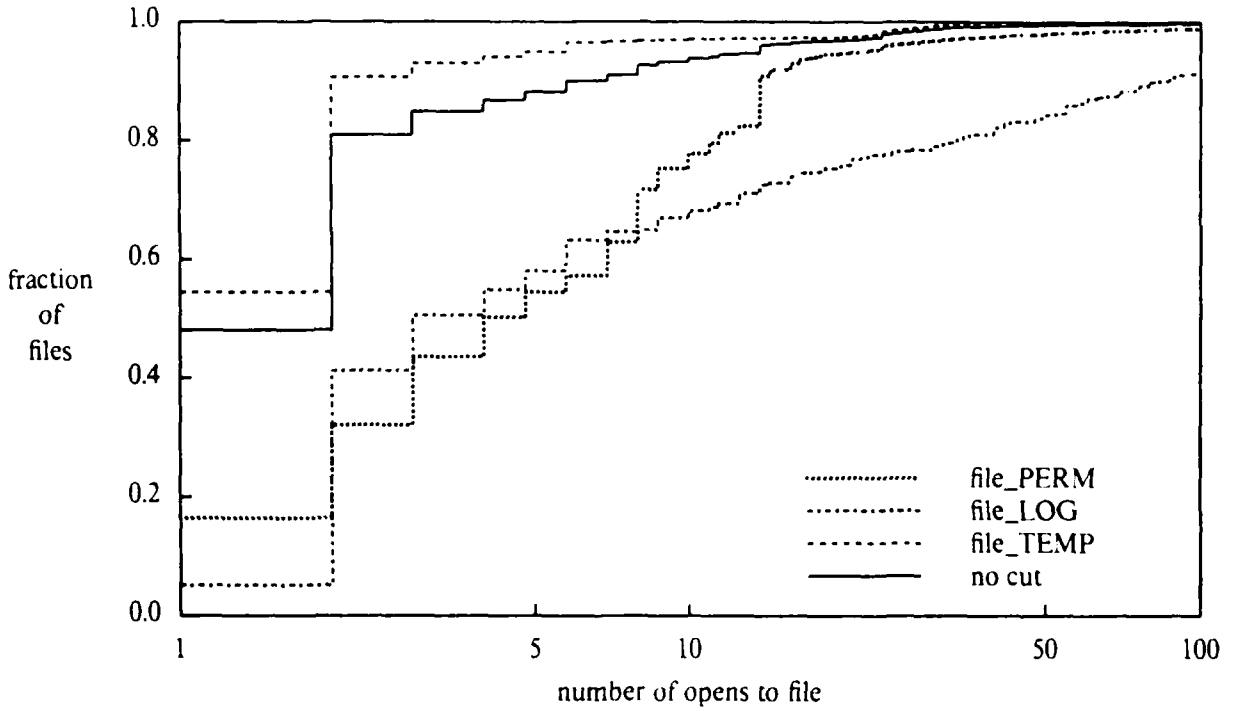


Figure B-23: Number of opens per active file (cumulative, file cut)

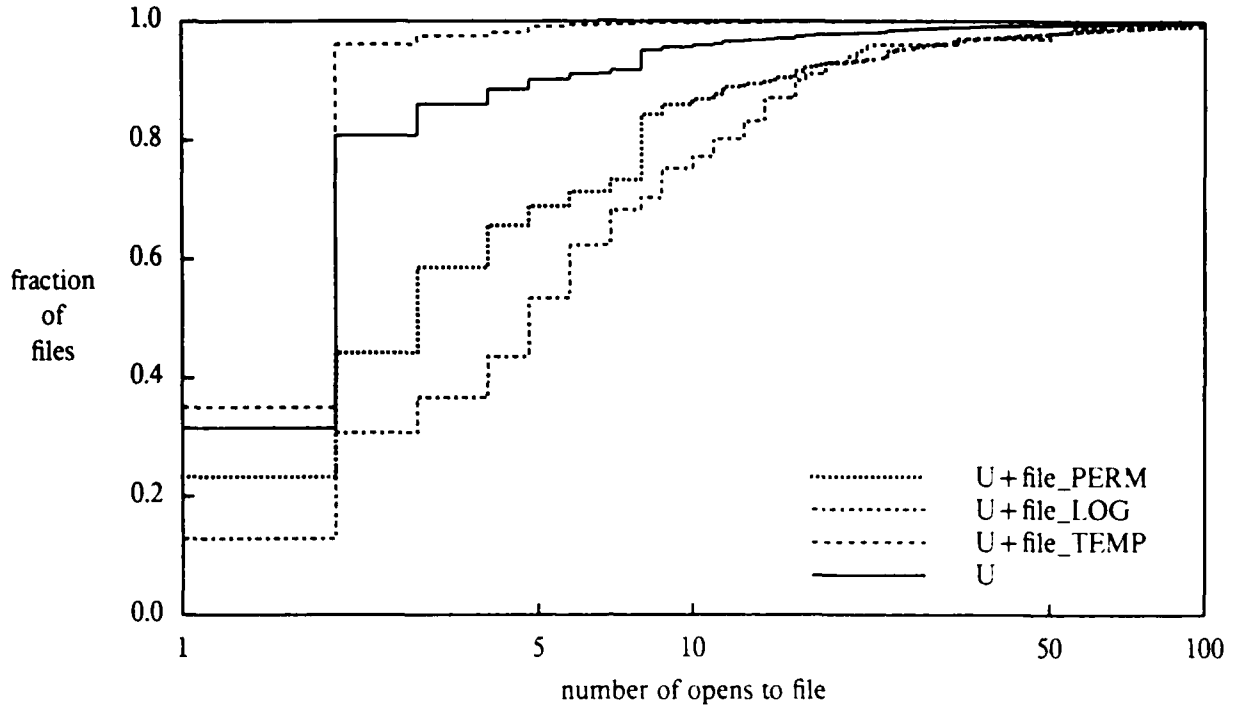


Figure B-24: Number of opens per active file (cumulative, U cut)

B.8. Time from File Open to Close

distribution	min	max	mean	median	std deviation
ruid_NET	0	1.2e4	6.9	0.14	86
ruid_SYSTEM	0	8.6e4	3.1	0.02	390
ruid_USER	0	7.6e4	30.4	0.35	520
owner_NET	0	1.3e4	9.6	0.14	115
owner_SYSTEM	0	8.6e4	6.1	0.05	402
owner_USER	0	4.8e4	37.1	0.35	568
file_LOG	0	8.6e4	33.4	0.08	1140
file_PERM	0	7.6e4	6.5	0.08	251
file_TEMP	0	4.8e4	20.5	0.22	335
U+file_LOG	0	5.9e3	14.5	0.29	226
U+file_PERM	0	4.8e4	15.8	0.32	438
U+file_TEMP	0	4.8e4	79.9	0.60	781
U	0	4.8e4	39.4	0.35	588
no cut	0	8.6e4	11.8	0.1	369

Table B-9: Open time (seconds)

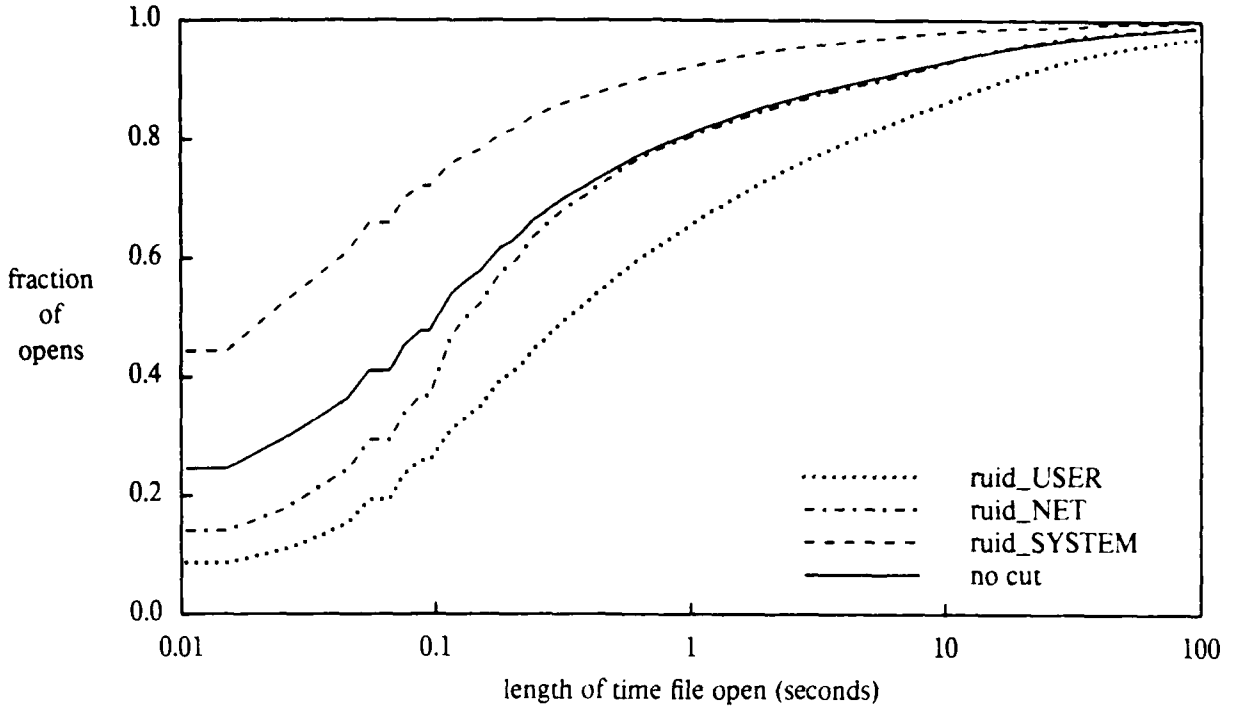


Figure B-25: Times from file open to close (cumulative, ruid cut)

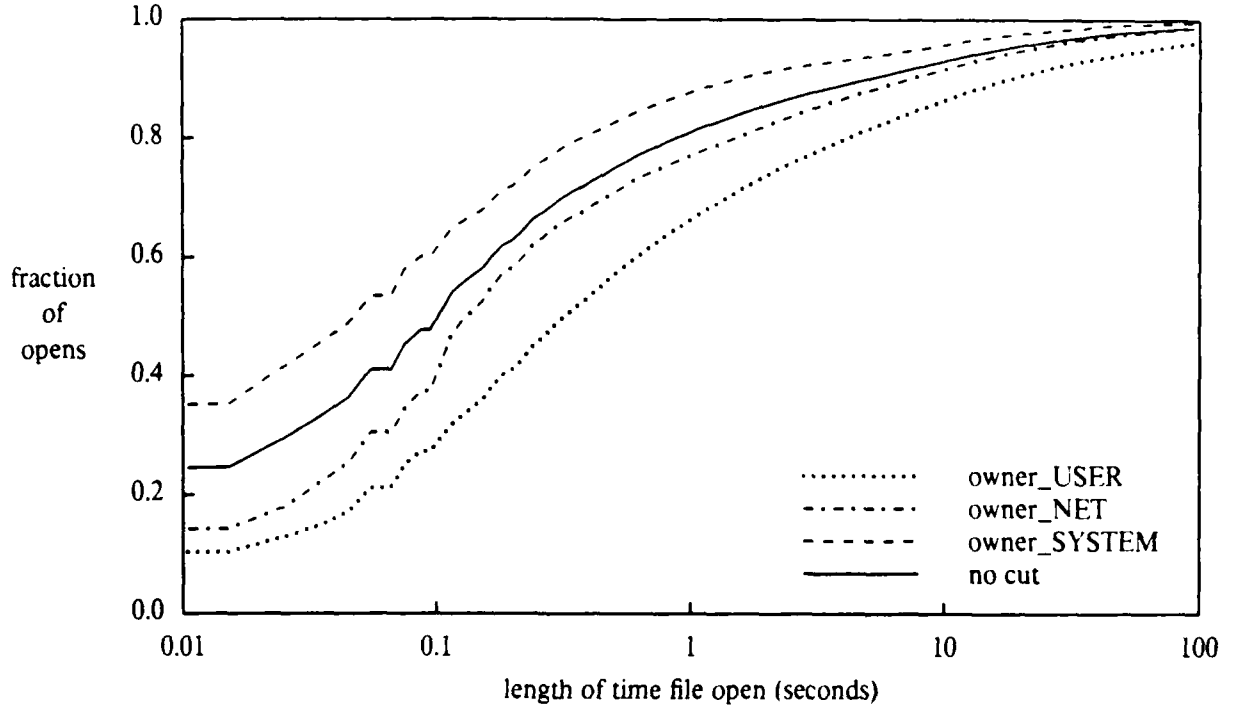


Figure B-26: Times from file open to close (cumulative, owner cut)

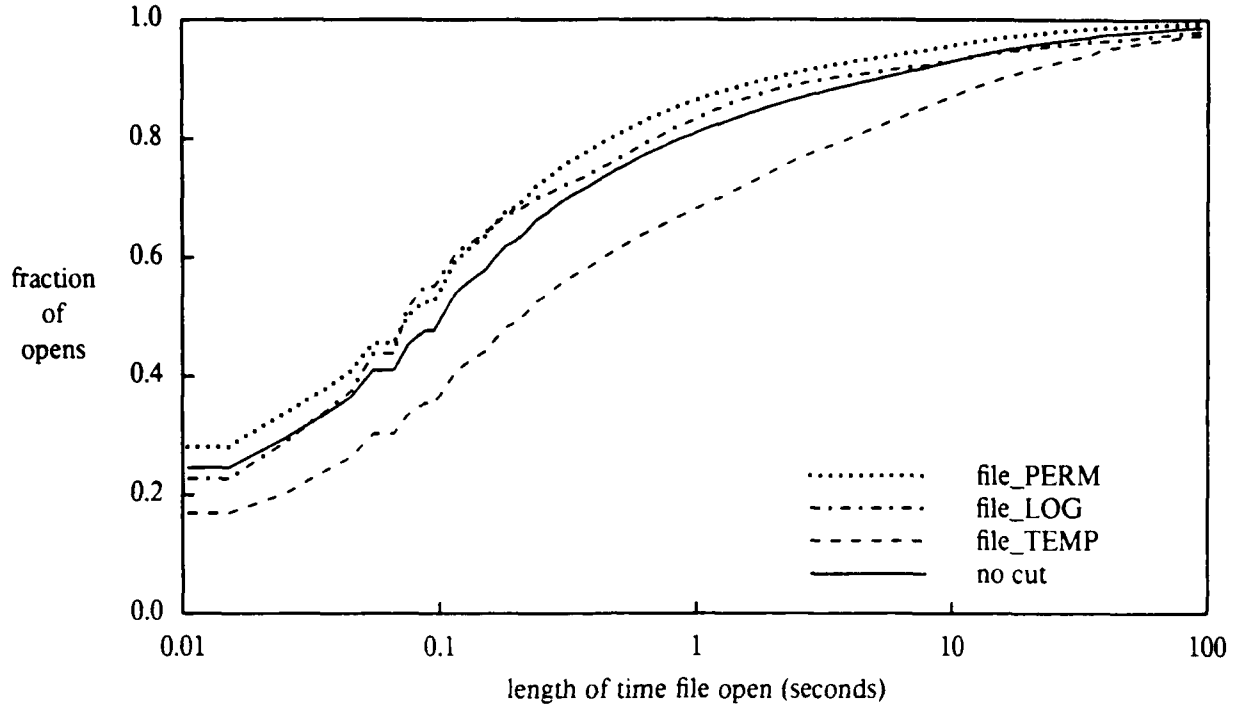


Figure B-27: Times from file open to close (cumulative, file cut)

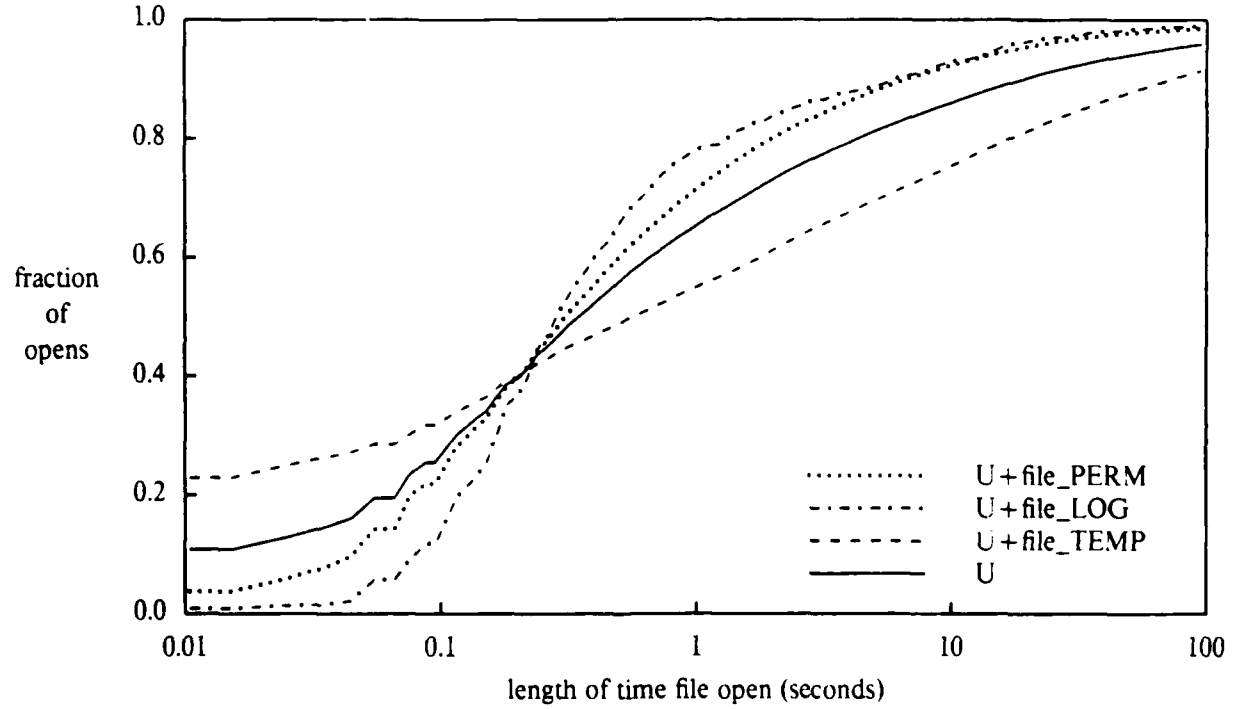


Figure B-28: Times from file open to close (cumulative, U cut)

B.9. File Interopen Intervals

distribution	min	max	mean	median	std deviation
ruid_NET	0	4.0e5	13640	13	2.9e4
ruid_SYSTEM	0	3.9e5	2816	60	1.1e4
ruid_USER	0	5.4e5	11520	105	3.5e4
owner_NET	0	4.0e5	13890	25	2.8e4
owner_SYSTEM	0	5.3e5	1740	60	9.1e3
owner_USER	0	5.4e5	17600	200	3.6e4
file_LOG	0	5.4e5	965	15	9.8e3
file_PERM	0	5.4e5	8215	60	2.4e4
file_TEMP	0	4.2e5	6655	3.6	1.8e4
U+file_LOG	0.02	5.4e5	31100	3100	5.7e5
U+file_PERM	0	5.4e5	21400	450	4.1e4
U+file_TEMP	0.01	4.2e5	1390	0.38	1.2e4
U	0	5.4e5	16900	120	3.7e4
no cut	0	5.4e5	7502	60	2.2e4

Table B-10: File interopen intervals (seconds)

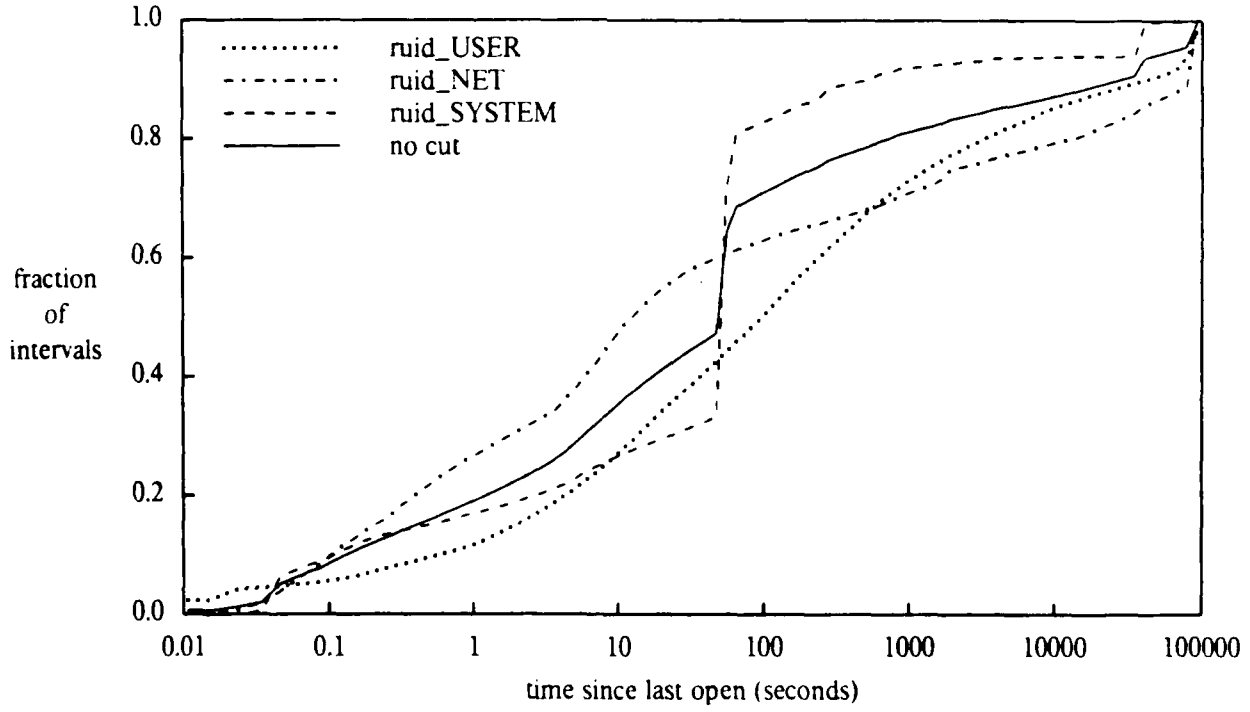


Figure B-29: File interopen intervals (cumulative, ruid cut)

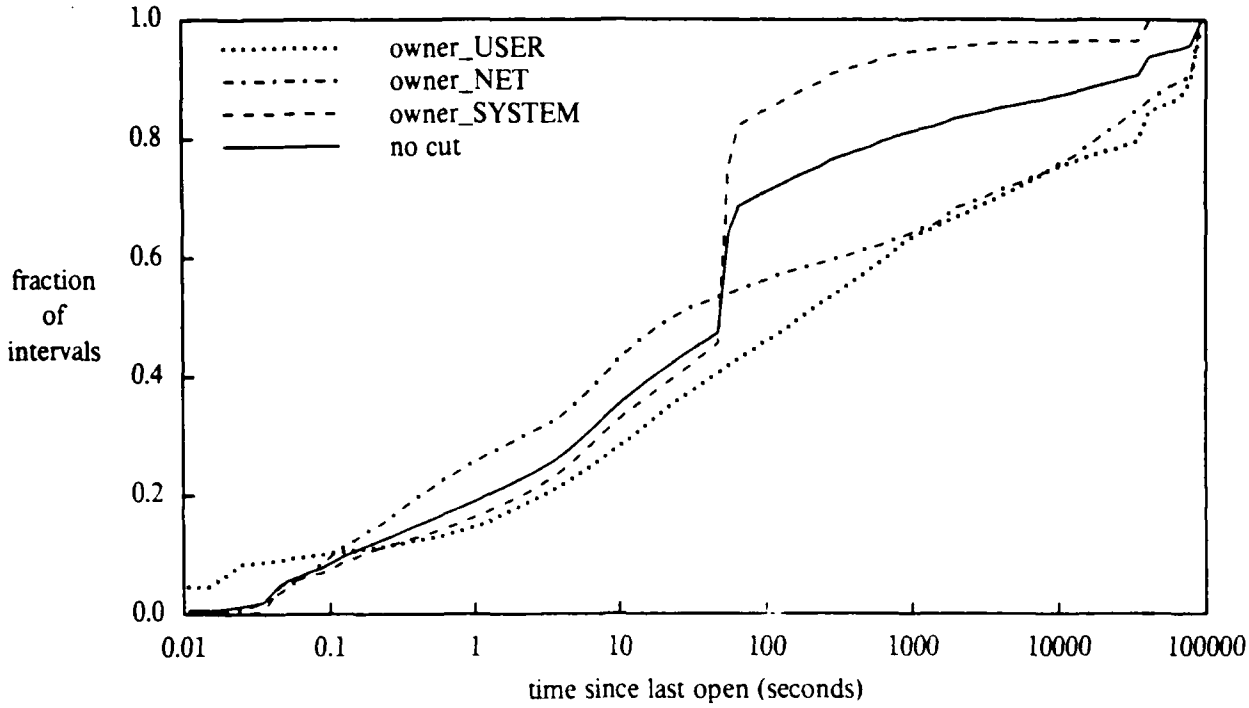


Figure B-30: File interopen intervals (cumulative, owner cut)

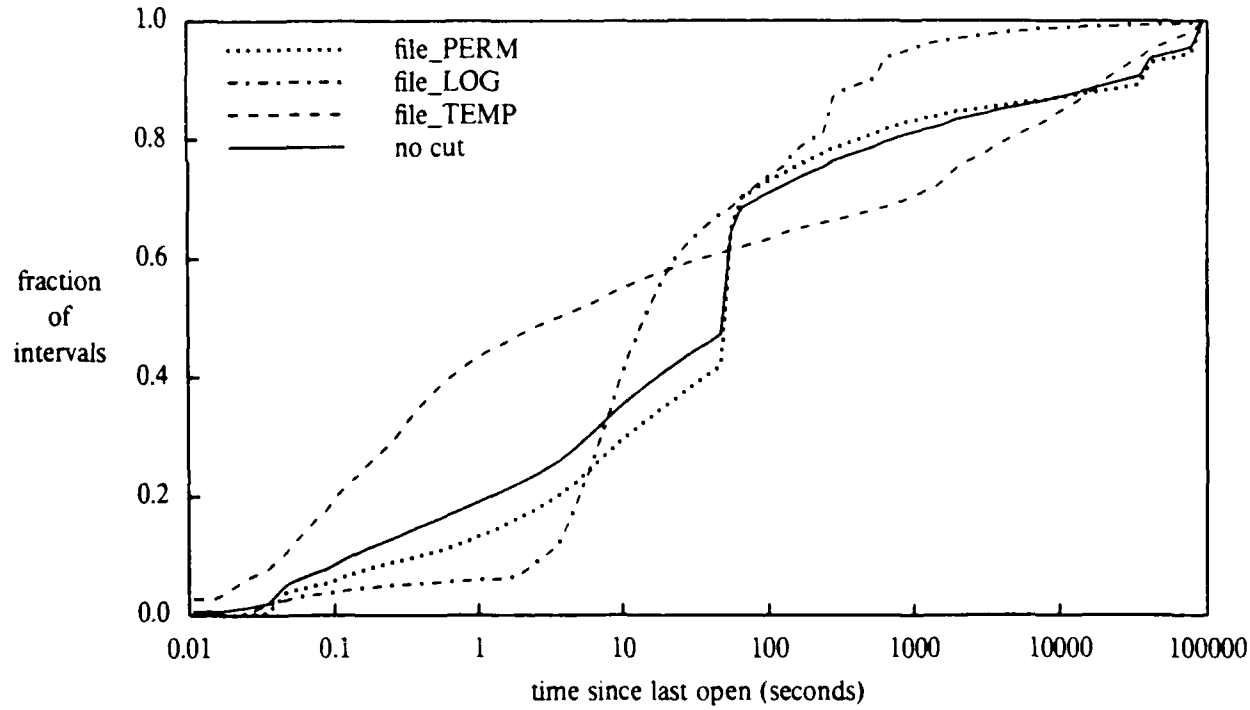


Figure B-31: File interopen intervals (cumulative, file cut)

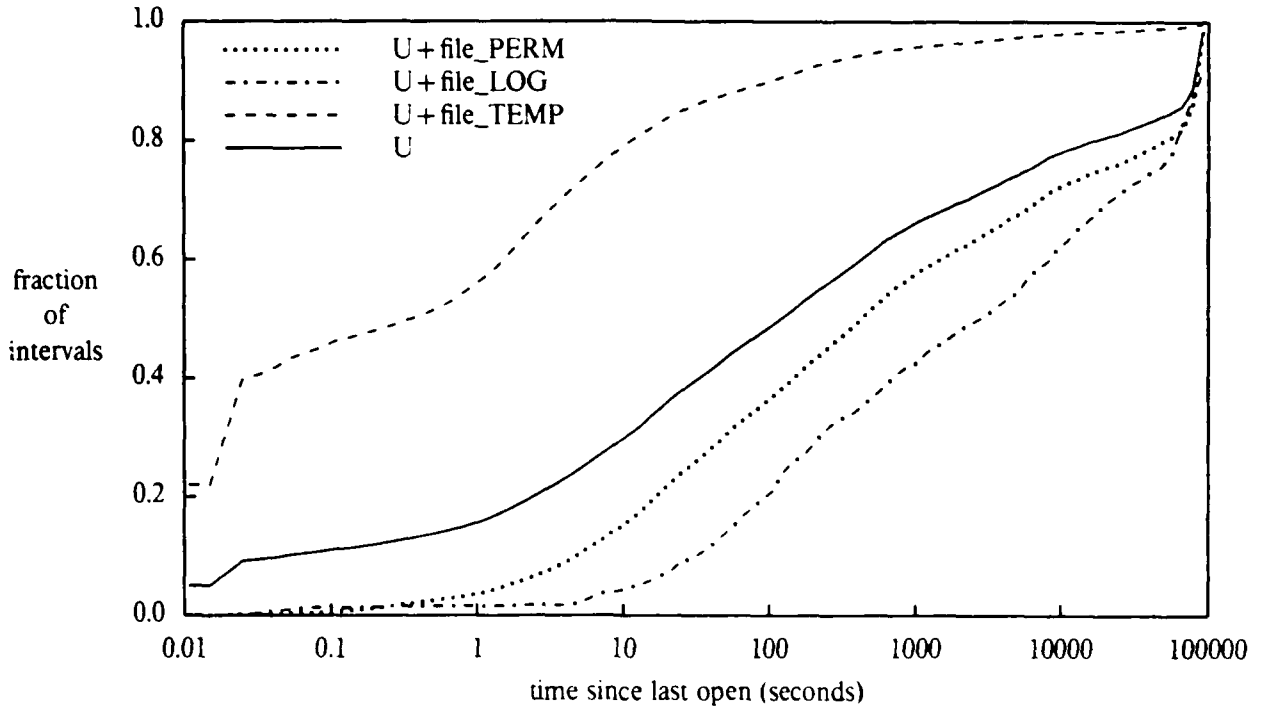


Figure B-32: File interopen intervals (cumulative, U cut)

B.10. File Lifetimes

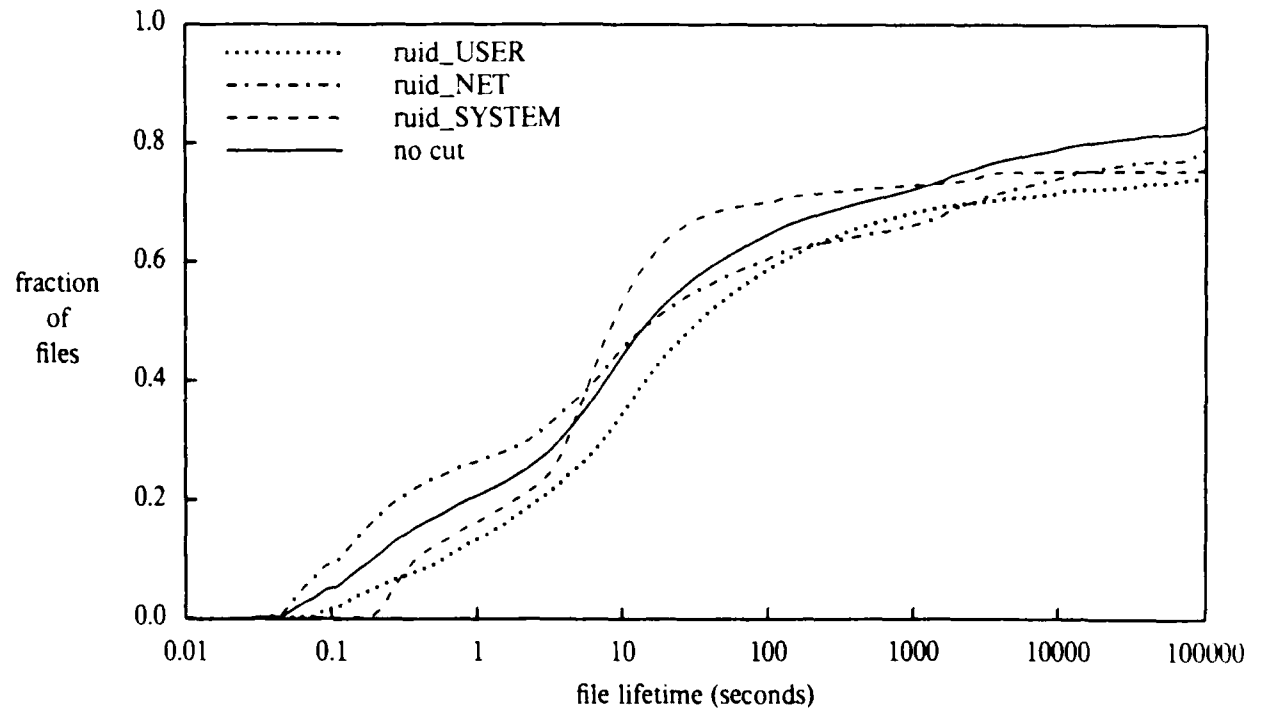


Figure B-33: File lifetimes (cumulative, files living beyond log period binned at right, ruid cut)

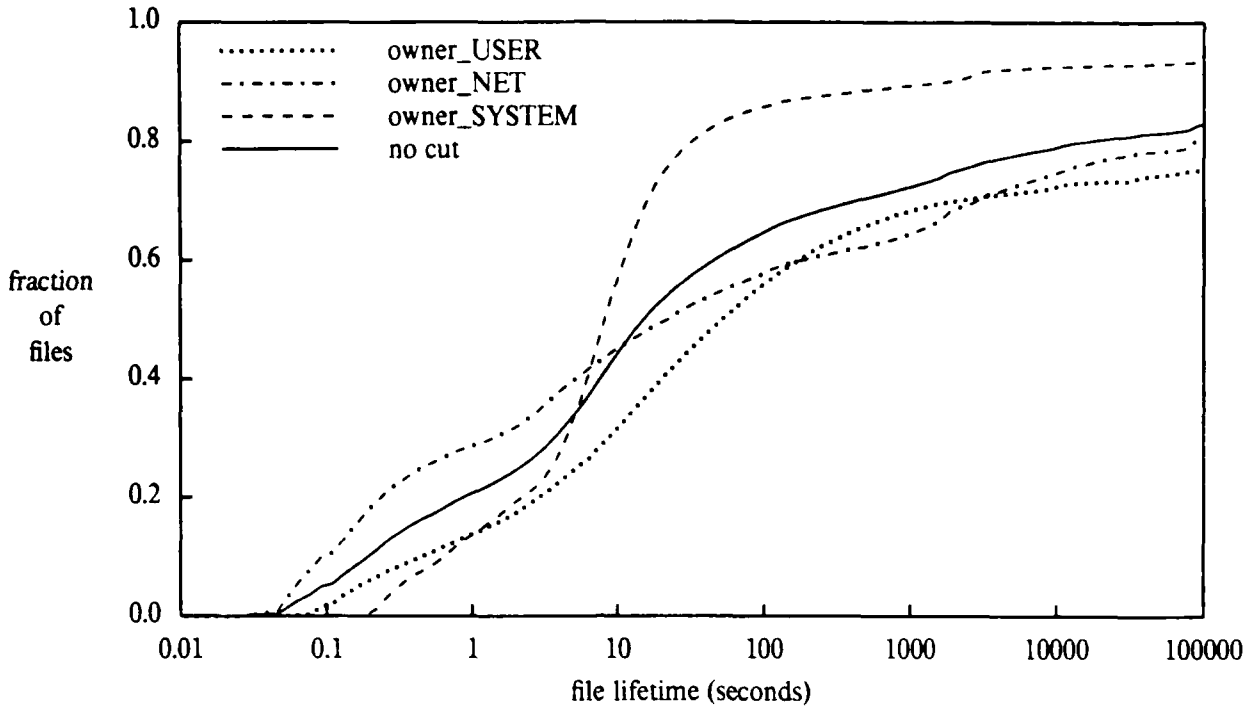


Figure B-34: File lifetimes (cumulative, files living beyond log period binned at right, owner cut)

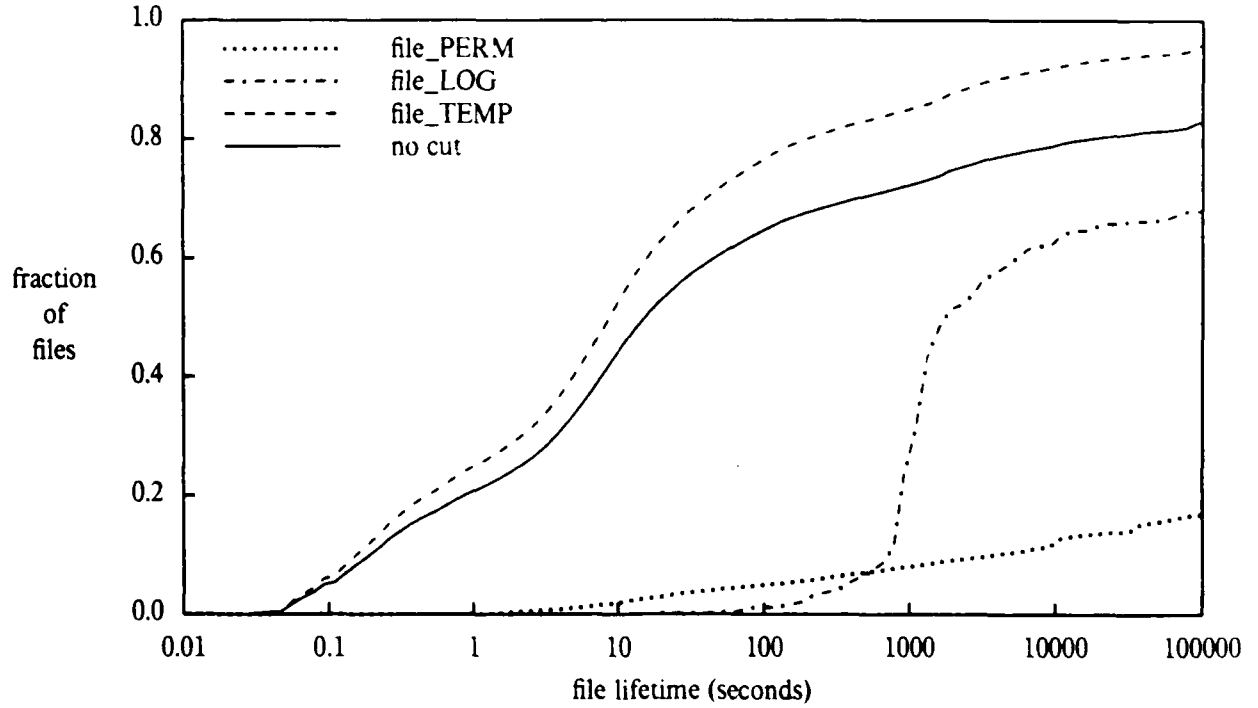


Figure B-35: File lifetimes (cumulative, files living beyond log period binned at right, file cut)

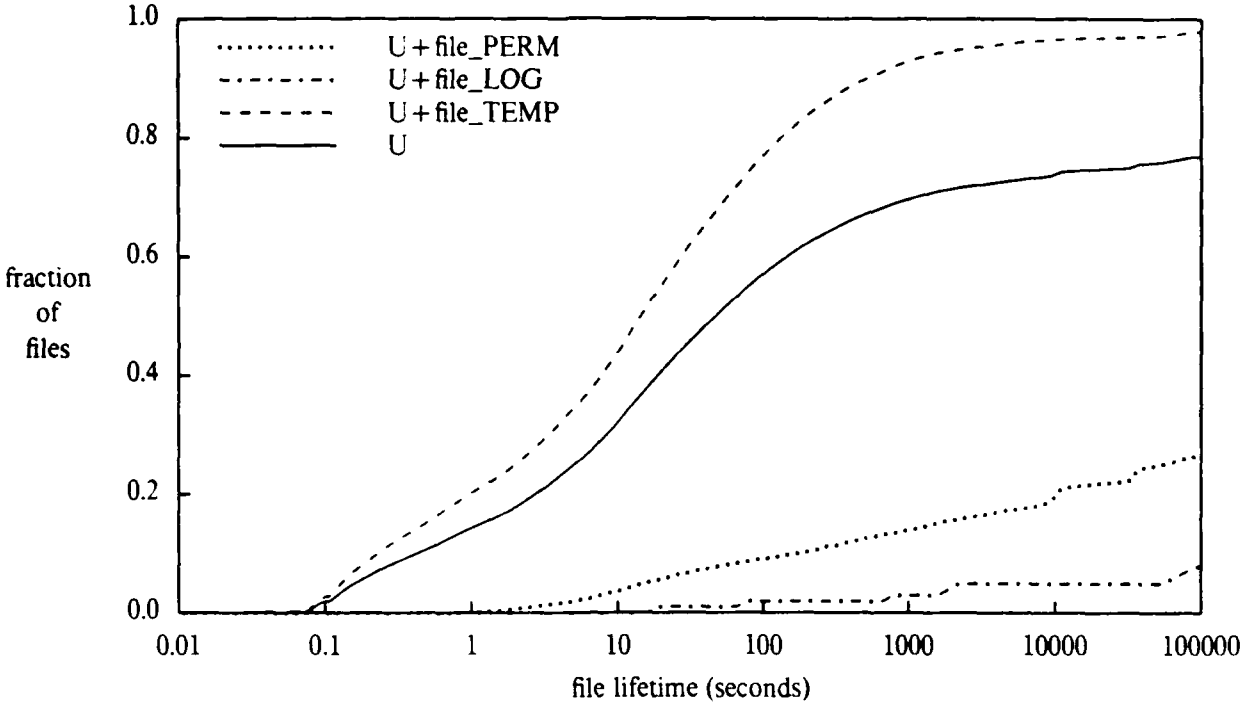


Figure B-36: File lifetimes (cumulative, files living beyond log period binned at right, U cut)

B.11. File Version Lifetimes

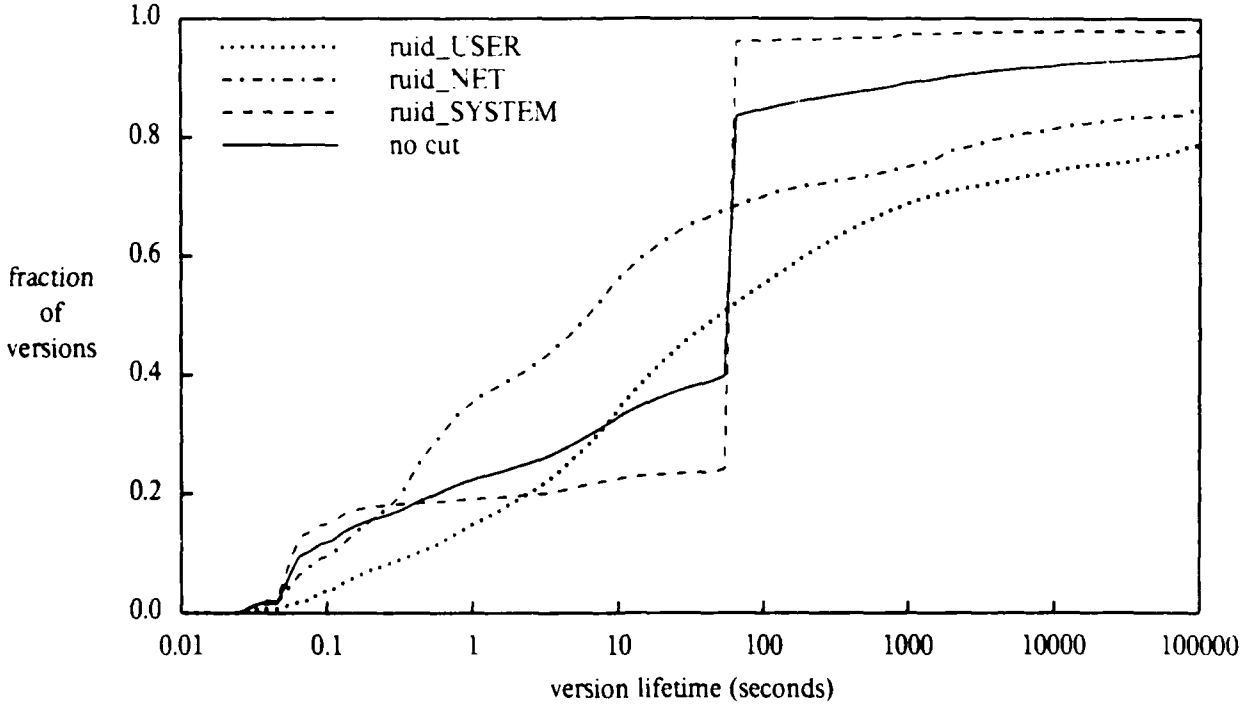


Figure B-37: Version lifetimes (cumulative, versions living beyond log period binned at right, ruid cut)

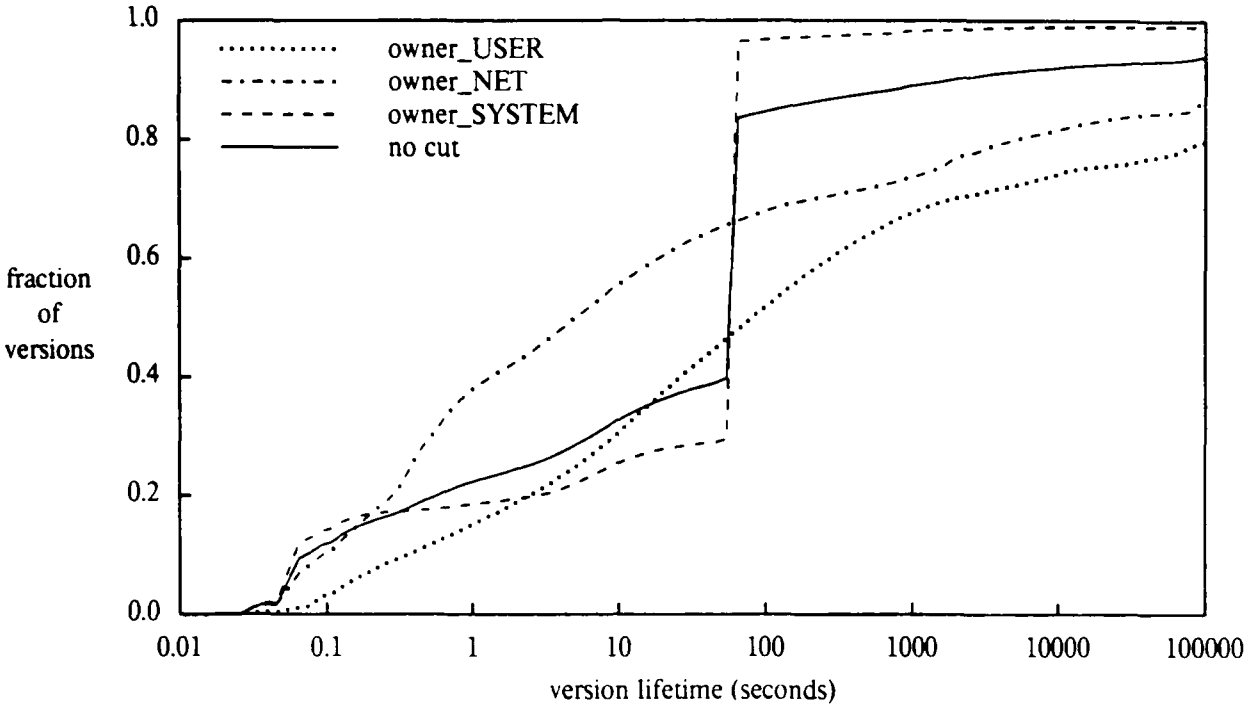


Figure B-38: Version lifetimes (cumulative, versions living beyond log period binned at right, owner cut)

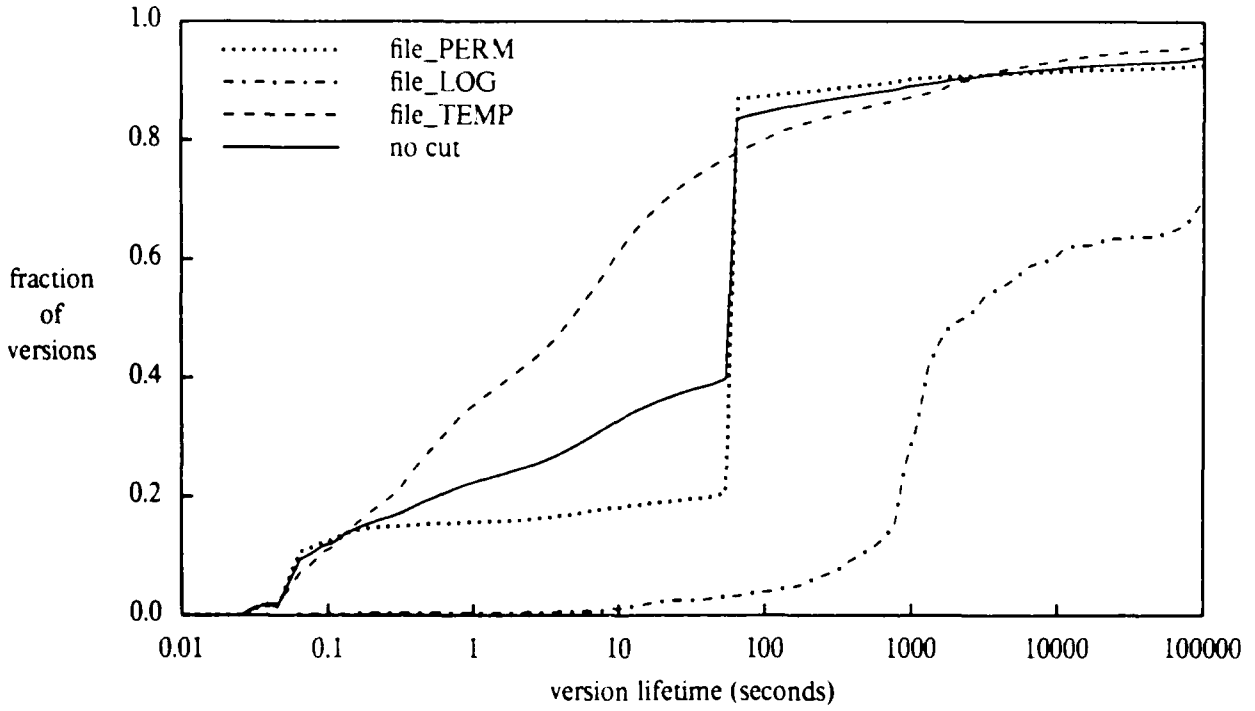


Figure B-39: Version lifetimes (cumulative, versions living beyond log period binned at right, file cut)

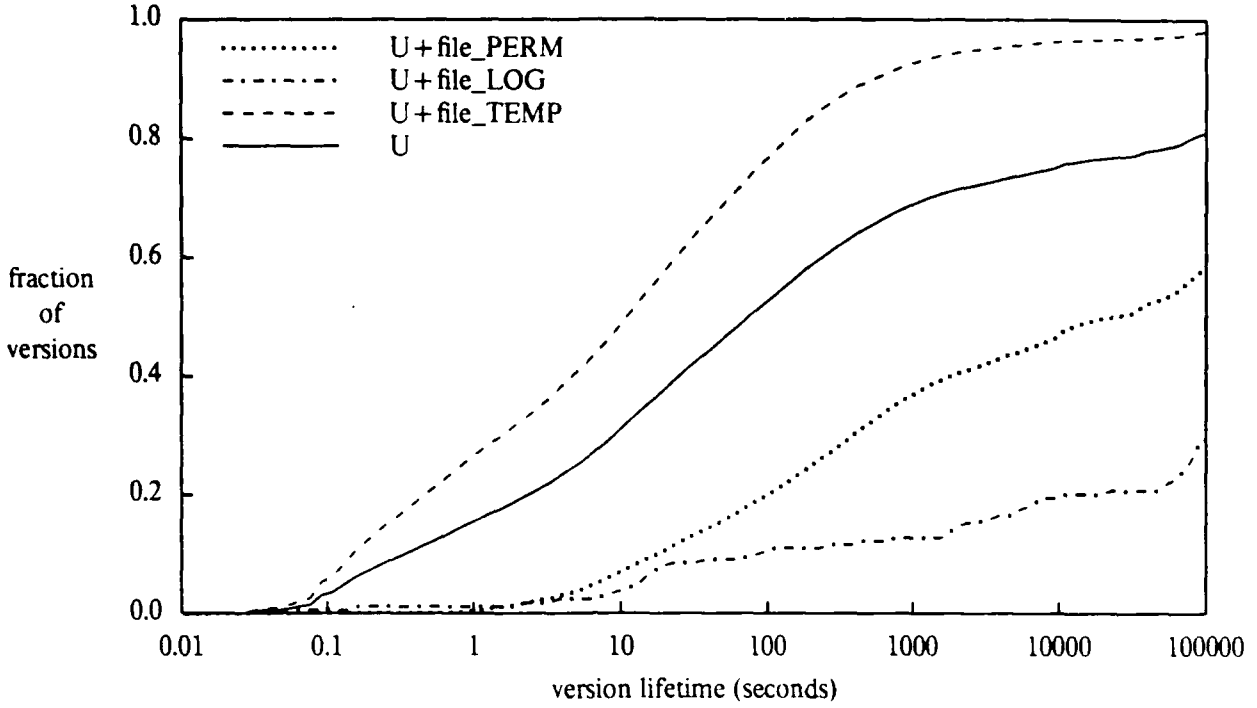


Figure B-40: Version lifetimes (cumulative, versions living beyond log period binned at right, U cut)

END

DTIC

9-86